Michael Gonzalez

CS 2302 Data Structures

Olac Fuentes

04/06/2019

Lab 5

For this lab, we have been tasked with implementing various methods regarding the use of Natural Language Processing. 1. Prompt the user to choose a table implementation 2. Read the file "glove.6B.50d.txt" and store each word and its embedding in a table with the chosen implementation, 3 Compute and display statistics describing your hash table.4. Return the similarity of words in the file, 5. Display running time if both options.

My solution for part 1 was simple. First, I would create a timer called start and store the current time in it, next it would print a few lines of information about the program. Here is where the user gets a choice and it stores that choice into a variable x, then it checks the contents of it. If x is an integer and a 1, it goes trough the program following the BST route. If it is an integer and a 2 it follows the Hash table route. Else, print “Wrong input” and end the program.

My solution for part 2 is the first 10 lines of code for the following functions. For BSTChoice lines 36-47 It creates an empty BST and then opens a .txt file using python’s native file reader and stores the file in variable f. next it goes into a for loop, for every line in f it does the following, split all lines by a space and saved it as a list in variable lines, next save the first element of that list as variable name, and an empty list called nums. In another for loop it appends every entry in lines to nums except for the first element. Next it stores name and nums inside another list p, and then inserts it into the BST until it reaches the end of the file. For HashChoice lines 85-94. It creates an empty Hash Table of size 11 and then opens a .txt file using python’s native file reader and stores the file in variable f. next it goes into a for loop, for every line in f it does the following, split all lines by a space and saved it as a list in variable lines, next save the first element of that list as variable name, and an empty list called nums. In another for loop it appends every entry in lines to nums except for the first element. Next it stores name and nums inside another list p, and then inserts it into the Hash table until it reaches the end of the file.

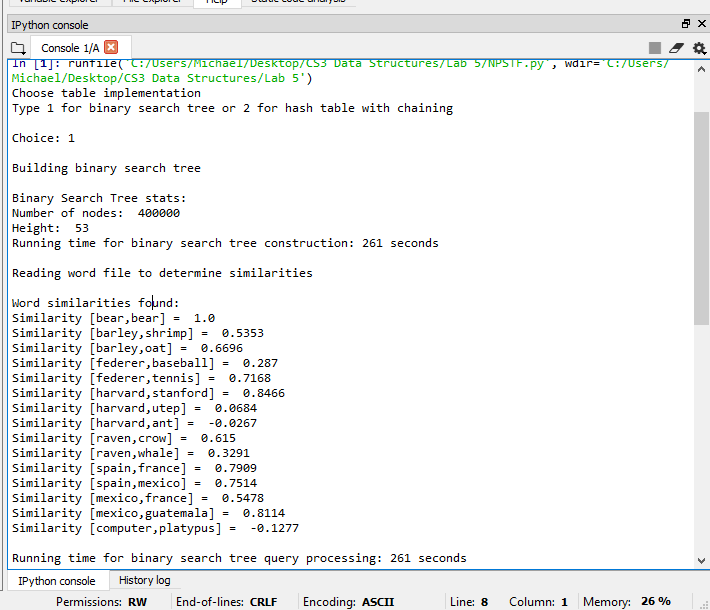
My solution for part 3 is the next 7 lines of code. In BSTChoice lines 48-56, it prints out the statistics of the BST such as, The number of nodes, the max depth and the resulting running time from its creation. NumNodes and maxDepth are methods created in my previous Lab assignment, Lab 3 Binary Search Trees, and the running time was calculated by subtracting the starting time minus the end time of the code at that point in time. For HashChoice lines 97-104, what is did was to print out statistics of the Hash table such as, Initial and final table size, Load factor, percemtage of empty lists, standard deviation of lengths of lists and running time for the Hash tables creation. Unfortunately, I was only able to implement the Load factor method by making a method called LoadFactor, which used a Hash table variable, H. it created a counter and then in a for loop for every I in H.item it added to count the length of the list in I and then returned count divided by the length of H.item.

My solution for part 4 was a to create a series of methods. DotProduct, Magnitude, SimH and SimT. DotProduct was a method that took two lists, e0 and e1, to calculate the dot product the method creates a variable to store the total called tot, and then in enters a for loop for as long as e0 is, since we are using the Glove.txt file we can be assured that the lengths of each list is the same throughout the code, next it takes each element in each list and multiplies it with one another then adding that value to tot. (Dot product formula is u\*v = u0\*v0+u1\*v1+: : :+un-1\*vn-1). Next, Magnuitude was fairly simple, I just returned the square root value of the dot product of the same list (Magnitude formula os |u| =sqrt(u\*u) = sqrt(DotProduct(ex,ex)). Lastly SimH and SimT are similar with the only difference being what implementation the user followed. First it found the list of numbers associated with the list and stored them in e0 and e1, next it calculated the dot product and saved it in a variable e0ne1, next it found each lists magnitude and saved it in a variable Me1 or Me0 and returned the result which was (e0ne1)/(Me0\*Me1).

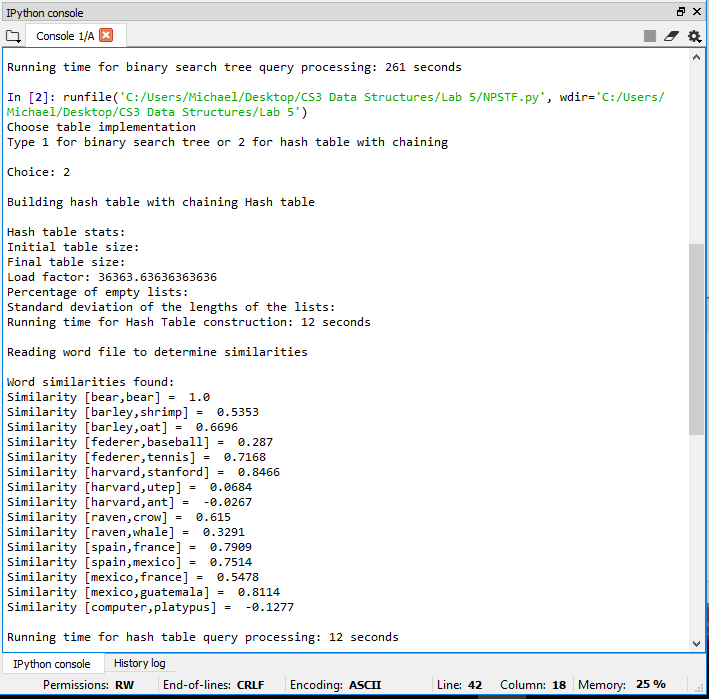
My solution for part 5 was to print out the resulting running time of the entire code. This was done by simply printing out the end time minus the starting time which resulted in the number of seconds that it took to complete the building of the list and the comparing the similarity of various words.

Experiments/Results

1.Sample Run 1:



2.Sample Run 2:



Conclusion

In conclusion, I learned how to program various methods about Reading and using txt files in my code, about Natural Language Processing, and how to calculate its running time. Aside from the methods, I have now a better appreciation of Natural Language Processing and its real-life application. I have become more comfortable with coding in python than in lab 4 and I believe that I will be able to learn more from future labs to come.

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.

– Michael Gonzalez

Appendix – code

#Author: Michael Gonzalez

#Course: CS 2302 Data Structures

#Lab 5

#TA: Anindita Nath

#Purpose:the purpose of this lab is to modify the code given in class and implement various methods.

import matplotlib.pyplot as plt

import numpy as np

import math

import time

class BST(object):

# Constructor

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

self.item = item

self.left = left

self.right = right

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=0):

self.item = []

self.num\_items = num\_items

if num\_items//size==1:

size = (size\*2)+1

for i in range(size):

self.item.append([])

def BSTChoice():

T = None

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

for line in f:

lines = line.split()

name = [lines[0]]

nums = []

for i in range(len(lines)-1):

nums.append(float(lines[i+1]))

p = [name,nums]

T = Insert(T,p)

print()

print("Binary Search Tree stats:")

print("Number of nodes: ",CountNodes(T))

print("Height: ",maxDepth(T))

elapsed\_time = time.time()-start

print("Running time for binary search tree construction:", round(elapsed\_time),"seconds")

print()

print("Reading word file to determine similarities")

print()

print("Word similarities found:")

print("Similarity [bear,bear] = ",round(simT('bear','bear',T),4))

print("Similarity [barley,shrimp] = ",round(simT('barley','shrimp',T),4))

print("Similarity [barley,oat] = ",round(simT('barley','oat',T),4))

print("Similarity [federer,baseball] = ",round(simT('federer','baseball',T),4))

print("Similarity [federer,tennis] = ",round(simT('federer','tennis',T),4))

print("Similarity [harvard,stanford] = ",round(simT('harvard','stanford',T),4))

print("Similarity [harvard,utep] = ",round(simT('harvard','utep',T),4))

print("Similarity [harvard,ant] = ",round(simT('harvard','ant',T),4))

print("Similarity [raven,crow] = ",round(simT('raven','crow',T),4))

print("Similarity [raven,whale] = ",round(simT('raven','whale',T),4))

print("Similarity [spain,france] = ",round(simT('spain','france',T),4))

print("Similarity [spain,mexico] = ",round(simT('spain','mexico',T),4))

print("Similarity [mexico,france] = ",round(simT('mexico','france',T),4))

print("Similarity [mexico,guatemala] = ",round(simT('mexico','guatemala',T),4))

print("Similarity [computer,platypus] = ",round(simT('computer','platypus',T),4))

print()

elapsed\_time2 = time.time()-start

print("Running time for binary search tree query processing:", round(elapsed\_time2),"seconds")

# data = np.array[file.readline().split()]

# for temp in data:

# print(temp)

#

# return

#

def HashChoice():

H = HashTableC(11)

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

for line in f:

lines = line.split()

name = [lines[0]]

nums = []

for i in range(len(lines)-1):

nums.append(float(lines[i+1]))

p = [name,nums]

InsertC(H,p,p[1])

print()

print("Hash table stats:")

print("Initial table size:")

print("Final table size:")

print("Load factor:",LoadFactor(H))

print("Percentage of empty lists:")

print("Standard deviation of the lengths of the lists:")

elapsed\_time = time.time()-start

print("Running time for Hash Table construction:", round(elapsed\_time),"seconds")

print()

print("Reading word file to determine similarities")

print()

print("Word similarities found:")

print("Similarity [bear,bear] = ",round(simH('bear','bear',H),4))

print("Similarity [barley,shrimp] = ",round(simH('barley','shrimp',H),4))

print("Similarity [barley,oat] = ",round(simH('barley','oat',H),4))

print("Similarity [federer,baseball] = ",round(simH('federer','baseball',H),4))

print("Similarity [federer,tennis] = ",round(simH('federer','tennis',H),4))

print("Similarity [harvard,stanford] = ",round(simH('harvard','stanford',H),4))

print("Similarity [harvard,utep] = ",round(simH('harvard','utep',H),4))

print("Similarity [harvard,ant] = ",round(simH('harvard','ant',H),4))

print("Similarity [raven,crow] = ",round(simH('raven','crow',H),4))

print("Similarity [raven,whale] = ",round(simH('raven','whale',H),4))

print("Similarity [spain,france] = ",round(simH('spain','france',H),4))

print("Similarity [spain,mexico] = ",round(simH('spain','mexico',H),4))

print("Similarity [mexico,france] = ",round(simH('mexico','france',H),4))

print("Similarity [mexico,guatemala] = ",round(simH('mexico','guatemala',H),4))

print("Similarity [computer,platypus] = ",round(simH('computer','platypus',H),4))

print()

elapsed\_time2 = time.time()-start

print("Running time for hash table query processing:", round(elapsed\_time2),"seconds")

def dotProduct(e0,e1):

tot = 0

for i in range(len(e0)):

tot += e0[i]\*e1[i]

return tot

def Magnitude(ex):

return math.sqrt(dotProduct(ex,ex))

def simH(w0,w1,H):

e0 = FindC(H,w0)

e1 = FindC(H,w1)

e0ne1 = dotProduct(e0,e1)

Me0 = Magnitude(e0)

Me1 = Magnitude(e1)

result = (e0ne1)/(Me0\*Me1)

return result

def simT(w0,w1,T):

e0 = Find(T,w0)

e1 = Find(T,w1)

e0ne1 = dotProduct(e0,e1)

Me0 = Magnitude(e0)

Me1 = Magnitude(e1)

result = (e0ne1)/(Me0\*Me1)

return result

#########################################################

# BST Code

def Insert(T,newItem):

if T == None:

T = BST(newItem)

elif T.item[0][0] > newItem[0][0]:

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

else:

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

return T

def Delete(T,del\_item):

if T is not None:

if del\_item < T.item:

T.left = Delete(T.left,del\_item)

elif del\_item > T.item:

T.right = Delete(T.right,del\_item)

else: # del\_item == T.item

if T.left is None and T.right is None: # T is a leaf, just remove it

T = None

elif T.left is None: # T has one child, replace it by existing child

T = T.right

elif T.right is None:

T = T.left

else: # T has two chldren. Replace T by its successor, delete successor

m = Smallest(T.right)

T.item = m.item

T.right = Delete(T.right,m.item)

return T

def InOrder(T):

# Prints items in BST in ascending order

if T is not None:

InOrder(T.left)

print(T.item[0][0],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[0][0])

InOrderD(T.left,space+' ')

def Smallest(T):

# Returns smallest item in BST. Error if T is None

if T.left is None:

return T

else:

return Smallest(T.left)

def CountNodes(T):

Count = 0

if T is not None:

Count += 1

else:

return 0

return Count + CountNodes(T.right) + CountNodes(T.left)

def maxDepth(T):

if T is None:

return 0 ;

else :

# Compute the depth of each subtree

lDepth = maxDepth(T.left)

rDepth = maxDepth(T.right)

# Use the larger one

if (lDepth > rDepth):

return lDepth+1

else:

return rDepth+1

def Find(T,k):

# Returns the address of k in BST, or None if k is not in the tree

if T is None or T.item[0][0] == k:

return T.item[1]

if T.item[0][0]<k:

return Find(T.right,k)

return Find(T.left,k)

#########################################################

#````````HASH Code

def InsertC(H,k,l):

# Inserts k in appropriate bucket (list)

# Does nothing if k is already in the table

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

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

H.num\_items += 1

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

return b, -1, -1

def h(s,n):

r = 0

for c in s:

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

return r

def LoadFactor(H):

count = 0.0

for i in H.item:

count +=len(i)

return count/len(H.item)

#########################################################

#

#H = HashTableC(11)

#A = ['data','structures','computer','science','university','of','texas','at','el','paso']

#for a in A:

# InsertC(H,a,len(a))

# print(H.item)

#

#for a in A: # Prints bucket, position in bucket, and word length

# print(a,FindC(H,a))

#########################################

# User Interface #

#########################################

start = time.time()

print("Choose table implementation")

print("Type 1 for binary search tree or 2 for hash table with chaining")

x =input("Choice: ")

print()

if int(x) == 1:

print("Building binary search tree")

BSTChoice()

elif int (x) == 2:

print("Building hash table with chaining Hash table")

HashChoice()

else:

print("Wrong input Try Again")