Lab 5 Report

Lab 5, the lab assigned this week, was based around the idea of searching for words in different data structures and how efficient these structures were. The two structures we had to work with were a Binary-Search Tree and a Hash Table. Each node in each of the structures was required to contain a word, and an embedding that represented the words as points position in a . The Two Words were compared to see how close they were, and it returned a floating point

In the first step, the method takes the file and stores each different line into a list after reading it. For the array with words comparisons, each index of the list contains only two words, for the other list each index contains the word and its embeddings.

In the second step, the lists with their data are split by the spaces in their lines. What this does is create a list of lists that contain all the elements of each index This is where the list containing two words stops, as there is no need for the next step.

From here the embeddings are separated from the words and put in a separate list in the [ index each having their own index as they are changed from strings to floats in order to be used in computations.

Hash Table output

In this portion of the lab, a Hash-Table is to be created using the words in the word and embedding list. Here is the output when reading the file for glove.txt

Hash Table Stats:

------------------------------------------------------------------

Intial Table Size: 11

Final Table Size: 393215

Load Factor: 0.8318375443459685

Percent of Empty Lists: 99.9173480157166 %

Standard Deviation of Length of Lists: 142.04472410698668

Time Taken To Build Hash Table: 21.538148880004883 Seconds

Word Similarities Found:

1 : ['the', '0.418', '0.24968', '-0.41242', '0.1217', '0.34527', '-0.044457', '-0.49688', '-0.17862', '-0.00066023', '-0.6566', '0.27843', '-0.14767', '-0.55677', '0.14658', '-0.0095095', '0.011658', '0.10204', '-0.12792', '-0.8443', '-0.12181', '-0.016801', '-0.33279', '-0.1552', '-0.23131', '-0.19181', '-1.8823', '-0.76746', '0.099051', '-0.42125', '-0.19526', '4.0071', '-0.18594', '-0.52287', '-0.31681', '0.00059213', '0.0074449', '0.17778', '-0.15897', '0.012041', '-0.054223', '-0.29871', '-0.15749', '-0.34758', '-0.045637', '-0.44251', '0.18785', '0.0027849', '-0.18411', '-0.11514', '-0.78581']

For this portion of the lab a binary search tree was to be created to find each element. The word and embedding set is sorted by its [0] element where the words are contained. This is done using the list.sort() method from Python. From here a method named listToTree is called with and empty tree and the list containing the words and embeddings. The method finds the middle of the list by finding len(list)//2, from here it checks if T is none and if so a new Binary-Search Tree is created from the middle element. From there the left node of the tree is recursively declared as calling the same tree and calling all word and embeddings before the middle element. The same is then done for the right node of the tree, only calling the word and embeddings after the middle of the list. The made tree is then returned.

BST Stats:

----------------------------------------------------------------------

Number of Nodes: 327091

Height Of Binary Tree: 19

Traceback (most recent call last):

Time Taken To Build BST: 19.439356088638306 Seconds

File "/Users/andymonreal/CS3files/Lab\_5/HashLab.py", line 385, in <module>

print(format(compareSimilarities(T, words[i][0], words[i][1])))

Reading Word File To Determine Similarities...

File "/Users/andymonreal/CS3files/Lab\_5/HashLab.py", line 119, in compareSimilarities

dp = dotProduct(first, second)

Word Similarities Found:

File "/Users/andymonreal/CS3files/Lab\_5/HashLab.py", line 131, in dotProduct

dp += first[1][i] \* second[1][i]

TypeError: 'int' object is not subscriptable

1 : ['the', '0.418', '0.24968', '-0.41242', '0.1217', '0.34527', '-0.044457', '-0.49688', '-0.17862', '-0.00066023', '-0.6566', '0.27843', '-0.14767', '-0.55677', '0.14658', '-0.0095095', '0.011658', '0.10204', '-0.12792', '-0.8443', '-0.12181', '-0.016801', '-0.33279', '-0.1552', '-0.23131', '-0.19181', '-1.8823', '-0.76746', '0.099051', '-0.42125', '-0.19526', '4.0071', '-0.18594', '-0.52287', '-0.31681', '0.00059213', '0.0074449', '0.17778', '-0.15897', '0.012041', '-0.054223', '-0.29871', '-0.15749', '-0.34758', '-0.045637', '-0.44251', '0.18785', '0.0027849', '-0.18411', '-0.11514', '-0.78581']

*"""  
Adrian Monreal  
olac fuentes  
lab 5  
"""***"""  
. Prompt the user to choose a table implementation (binary search tree or hash table with chaining).  
2. Read the file ”glove.6B.50d.txt” and store each word and its embedding in a table with the chosen  
implementation. Each node in the BST or hash table must consist of a list of size two, containing the   
word (a string) and the embedding (a numpy array of length 50). For the hash table,   
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.   
Caution: do NOT recompute the load factor every time an item is entered to the table, instead,   
add a num items fields to your hash table class.  
3. Compute and display statistics describing your hash table.   
See the appendix for examples for both implementations. Feel free to suggest others.  
4. Read another file containing pairs of words (two words per line) and   
for every pair of words find and display the ”similarity” of the words.   
To compute the similarity between words w0 and w1, with embeddings e0 and e1, we use the cosine distance,  
 which ranges from -1 (very different) to 1 (very similar), given by:  
sim(w0,w1)= e0·e1 |e0 ||e1 |  
 where e0 · e1 is the dot product of e0 and e1 and |e0| and |e1| are the magnitudes of e0 and e1.  
Recall that the dot product of vectors u and v of length n is given by u·v = u0 ∗v0 +u1 ∗v1 +. . .+un−1 ∗vn−1  
and the magnitude of a vector u of length n is given by |u| = √u · u = 􏰀u20 + u21 + . . . u2n−1.  
 5. Display the running times required to build the table (item 2) and to compute the similarities (item 4).   
 Do not include the time required for displaying results.   
 Use a large enough word file for item 4 in order to derive meaningful results.  
"""**# Implementation of hash tables with chaining using strings  
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  
  
  
def Insert(T, newItem):  
 if T == None:  
 T = BST(newItem)  
 else:  
 if T.item > newItem:  
 T.left = Insert(T.left, newItem)  
 else:  
 T.right = Insert(T.right, newItem)  
 return T  
  
  
# Returns position in alphabet  
def char\_position(letter):  
 return ord(letter) - 97  
  
  
# Returns true if item is to the left and false if it is to the right  
def compareWords(w1, w2):  
 if w1 > w2:  
 return True  
 return False  
  
  
# Converts List of sorted words into A Tree  
def listToTree(T, A):  
 if len(A) == 0:  
 return None  
 mid = len(A) // 2  
 if T is None:  
 head = BST(A[mid])  
 head.left = listToTree(T, A[:mid])  
 head.right = listToTree(T, A[mid + 1:])  
 return head  
  
  
# Returns the number of nodes in Given Tree  
def numNodes(T):  
 if T is None:  
 return 0  
 return 1 + numNodes(T.left) + numNodes(T.right)  
  
  
# Returns Height of Given Tree  
def getHeight(T):  
 if T is None:  
 return 0  
 else:  
 leftHeight = getHeight(T.left)  
 rightHeight = getHeight(T.right)  
 if leftHeight > rightHeight:  
 return leftHeight + 1  
 else:  
 return rightHeight + 1  
  
  
# Finds A Node In A BST  
def findB(T, k):  
 if T is None:  
 return -1  
 if T.item[0] == k:  
 return T.item  
 cur = T.item[0]  
 if compareWords(cur, k):  
 return findB(T.left, k)  
 return findB(T.right, k)  
  
  
# Calculates The Similarites Of Two Words In A Binary Search Tree  
def compareSimilarities(T, first, second):  
 # Declaring Two words To Work With  
 first = findB(T, first)  
 second = findB(T, second)  
 # Finding Dot Product of Word Embeddings  
 dp = dotProduct(first, second)  
 # Finding Magnitudes of two words  
 mag1 = MagOfWord(first)  
 mag2 = MagOfWord(second)  
 denom = mag1 \* mag2  
 return dp / denom  
  
  
# Gets The Dot Product of Two Words from a Binary-Tree  
def dotProduct(first, second):  
 dp = 0  
 for i in range(len(first[1])):  
 dp += first[1][i] \* second[1][i]  
 return dp  
  
  
# Calculates the Magnitude Of A Word From A Binary-Tree  
def MagOfWord(word):  
 m = 0  
 for i in range(len(word[1])):  
 m += word[1][i] \* word[1][i]  
 return math.sqrt(m)  
  
  
# Functions concerning hash tables  
# ------------------------------------------------------------------------------  
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 = []  
 self.size = size  
 self.num\_Items = 0  
 for i in range(size):  
 self.item.append([])  
  
  
def NumItems(H):  
 count = 0  
 for i in range(len(H.item)):  
 count += len(H.item[i])  
 return count  
  
  
def LoadFac(H):  
 count = 0  
 for i in range(len(H.item)):  
 count += len(H.item[i])  
 num\_Items = count  
 return num\_Items / len(H.item)  
  
  
def InsertC(H, k, e):  
 # 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([e])  
  
  
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] == k:  
 return H.item[b][i]  
 return -1  
  
  
def h(s, n):  
 r = 0  
 for c in s:  
 r = (r \* n + ord(c)) % n  
 return r  
  
  
# Turns list with words and embeddings into a Hash-Table  
def createHashTbl(A):  
 H = HashTableC(11, 0)  
 for i in range(len(A)):  
 # inserting elements into Hash-Table  
 elem = [A[i][0], A[i][1]]  
 InsertC(H, elem[0], elem)  
 H.num\_Items += 1  
 # Checks if load factor is = 1 and if so makes list size larger by \*2+1  
 if (H.num\_Items == H.size):  
 for i in range(H.size + 1):  
 H.item.append([])  
 H.size = (H.size \* 2) + 1  
  
 return H  
  
  
# Gets The Dot Product of Two Words in A Hash-Table  
def dotProdHash(first, second):  
 dp = 0  
 for i in range(len(first[1])):  
 dp += first[1][i] \* second[1][i]  
 return dp  
  
  
# Calculates the Magnitude Of A Word In A Hash-Table  
def MagOfWordHashTable(word):  
 mag = 0  
 for i in range(len(word[1])):  
 mag += word[1][i] \* word[1][i]  
 return math.sqrt(mag)  
  
  
# Compares Similarities In A Hash-Table  
def compareSimilaritiesH(H, first, second):  
 # Declaring Two Words  
 first = FindC(H, first)  
 second = FindC(H, second)  
 # FInding Dot Product Of Words Embeddings  
 dp = dotProdHash(first, second)  
 # Returning Magnitudes of Words  
 mag1 = MagOfWordHashTable(first)  
 mag2 = MagOfWordHashTable(second)  
 denom = mag1 \* mag2  
 return dp / denom  
  
  
# Returns the standard deviation in a Hash-Table  
def standDevH(H):  
 count = 0  
 # Summing length of lists  
 for i in H.item:  
 count += len(i)  
 avg = count / len(H.item)  
 count = 0  
 for i in H.item:  
 count += (len(i) - avg) \* (len(i) - avg)  
 avg = count / len(H.item)  
 return math.sqrt(avg)  
  
  
# Returns percent of empty lists  
def percentEmpty(H):  
 count = 0  
 for i in H.item:  
 if len(i) == 0:  
 count += 1  
 return (count \* 100) / len(H.item)  
  
  
# -------------------------------------------------------------------------------  
# Functions that read the file and make them into nodes  
# Converts the given file to an array of each line  
def fileToArray(filename):  
 file = open(filename, encoding=**"utf8"**)  
 A = file.readlines()  
 file.close  
 return A  
  
  
# Splits the elements of the string list into individuals  
def splitList(A):  
 B = []  
 for i in range(len(A)):  
 sp = A[i].split()  
 B.append(sp)  
 return B  
  
  
# Creates a list with the string word element in one field (Word) and an float array in the second (Embedding)  
def wordEmbedding(A):  
 B = []  
 for i in range(len(A)):  
 if A[i][0].isalpha():  
 # Putting float elements in one list  
 ls = np.array(A[i][1:])  
 # Changing the elements from strings to float points  
 lsr = ls.astype(np.float)  
 lis = [A[i][0], lsr]  
 B.append(lis)  
 return B  
**"""  
Main - -------------------------------------------------------------------------  
"""**print()  
print(**"Hash Table or Binary Search Tree"**)  
  
  
txt = input(**"Choice: "**)  
txt = str(txt)  
if txt == **'Hash Table'**:  
 words = fileToArray(**'glove.6B.50d.txt'**)  
 words = splitList(words)  
  
 print()  
 print(**"Creating Hash Table..."**)  
 print()  
  
 start = time.time()  
 filename = **'glove.6B.50d.txt'** inArr = fileToArray(filename)  
 splitArr = splitList(inArr)  
 word = wordEmbedding(splitArr)  
 hTable = createHashTbl(word)  
 end = time.time()  
  
 # Stats Of The Hash-Table  
 print(**"Hash Table Stats: "**)  
 print(**"------------------------------------------------------------------"**)  
 print(**"Intial Table Size: "**, 11)  
 print(**"Final Table Size: "**, len(hTable.item))  
 print(**"Load Factor: "**, LoadFac(hTable))  
 print(**"Percent of Empty Lists: "**, percentEmpty(hTable), **'%'**)  
 print(**"Standard Deviation of Length of Lists: "**, standDevH(hTable))  
 print(**"Time Taken To Build Hash Table: "**, abs(start - end), **" Seconds"**)  
 print()  
 print(**"Reading Word File To Determine Similarities..."**)  
 print()  
 print(**"Word Similarities Found:"**)  
  
 start = time.time()  
 # Displaying Words and Finding Similarities  
 for i in range(len(words)):  
 print(i + 1, **': '**, words[i], end=**''**)  
 print(**" = "**, end=**''**)  
 print(format(compareSimilaritiesH(hTable, words[i][0], words[i][1])))  
 end = time.time()  
  
 print()  
 print(**"Time Taken To Find 60 Similarites: "**, abs(start - end), **" Seconds"**)  
  
  
  
if txt == **"Binary Search Tree"**:  
 words = fileToArray(**'glove.6B.50d.txt'**)  
 words = splitList(words)  
  
 print()  
 print(**"Building Binary Search Tree..."**)  
 print()  
  
 start = time.time()  
 T = None  
 filename = **'glove.6B.50d.txt'** inArr = fileToArray(filename)  
 splitArr = splitList(inArr)  
 word = wordEmbedding(splitArr)  
 # Sorting words from txt file  
 word.sort()  
 T = listToTree(T, word)  
 end = time.time()  
  
 # Stats of the BST  
 print(**"BST Stats:"**)  
 print(**"----------------------------------------------------------------------"**)  
 print(**"Number of Nodes: "**, numNodes(T))  
 print(**"Height Of Binary Tree: "**, getHeight(T))  
 print(**"Time Taken To Build BST: "**, end - start, **" Seconds"**)  
 print()  
 print(**"Reading Word File To Determine Similarities..."**)  
 print()  
 print(**"Word Similarities Found:"**)  
  
 start = time.time()  
 # Printing Words and Displaying Similarities  
 for i in range(len(words)):  
 print(i + 1, **': '**, words[i], end=**''**)  
 print(format(compareSimilarities(T, words[i][0], words[i][1])))  
 end = time.time()  
  
 print( abs(start - end), **" Seconds"**)  
  
else:  
 print(**"make sure to copy as expected"**)

I Adrian Monreal 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.