Erick Perchez

80582912

CS 2302 Data Structures

4/14/2019

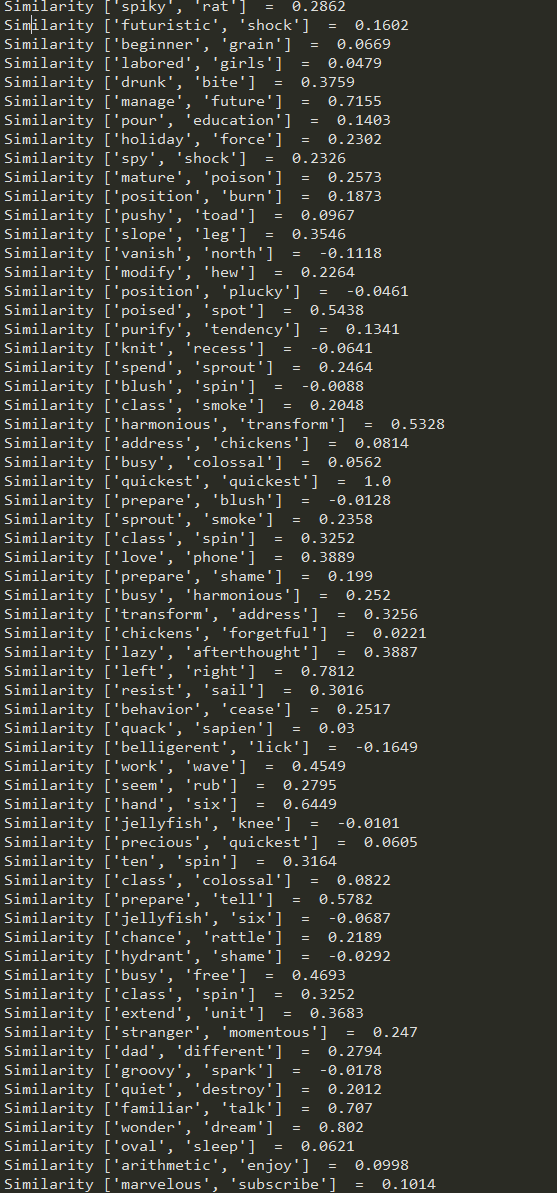
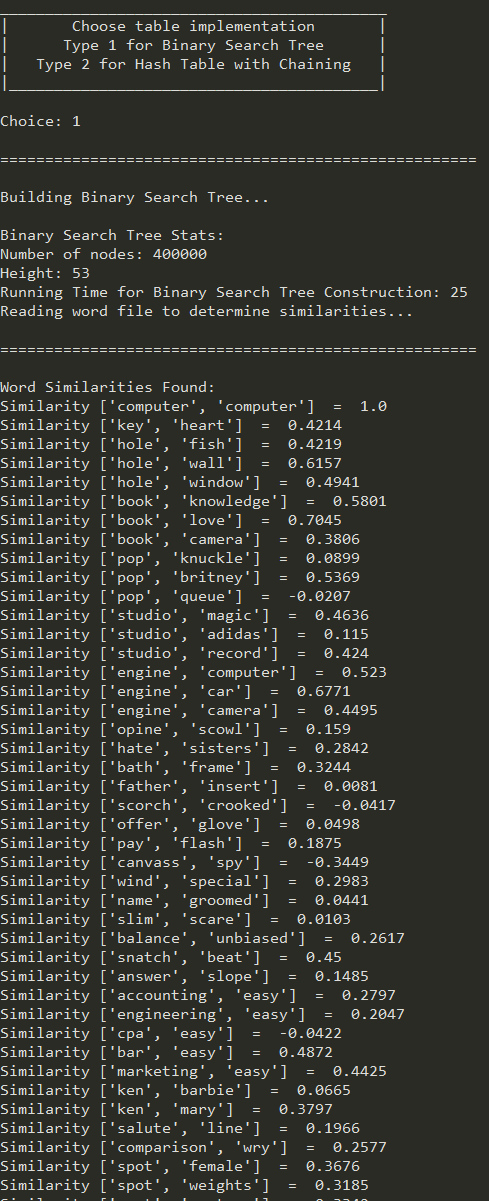
Lab 5

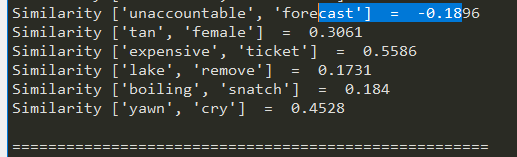
**Introduction:** In this lab we were given one file gave us 400,000 lines of words and their embedding. We were to use those words and embeddings to find similarities between words we wrote on a different list. We were to do this using hash tables and binary search trees.

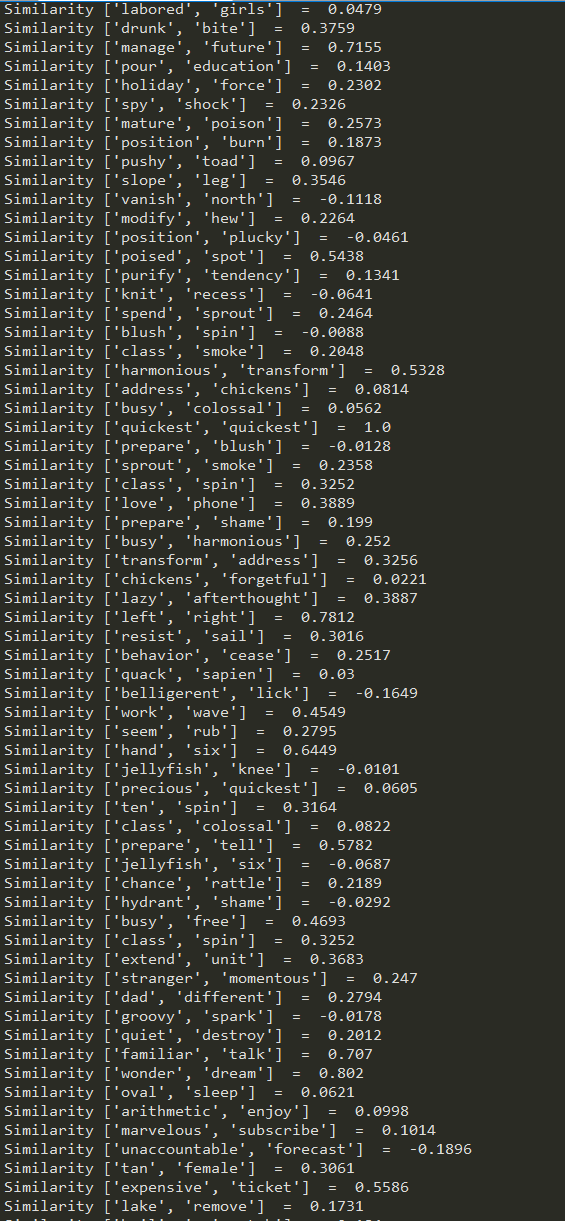
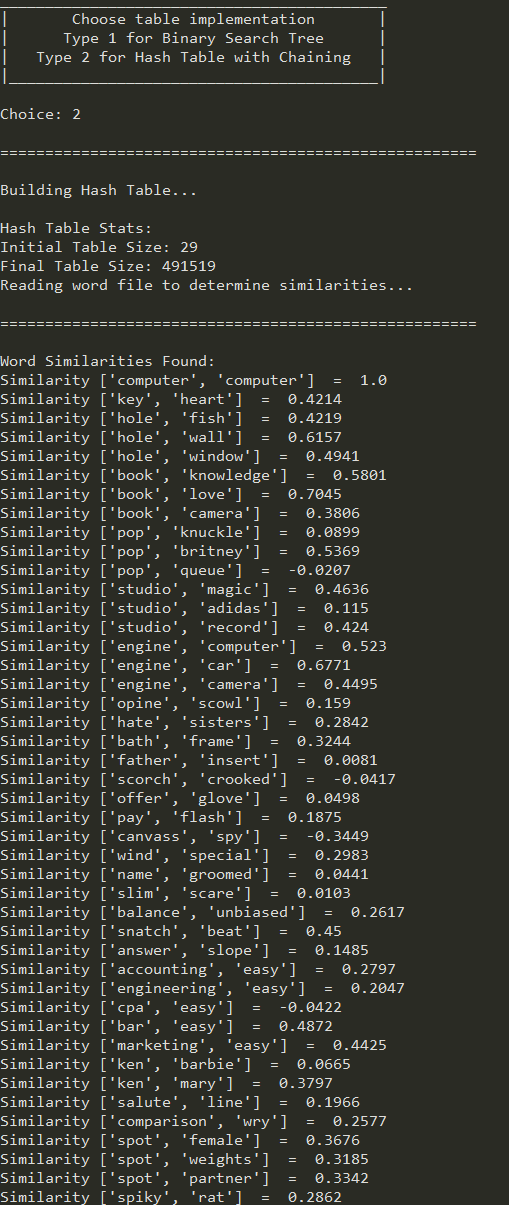
**Proposed solution:**

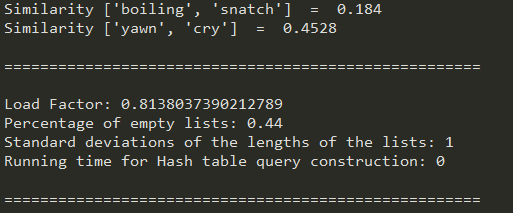
1. Prompt the user to choose a table implementation- I simply asked the user to choose their preferred implementation. By choosing 1, they would pick BST, and by choosing 2 they would pick hash table implementation. Running Time: O(1)
2. Read file “glove.6B.50d.txt” and store in the desired implementation- I would have had the file open from the beginning of the code and it was time to read it. If the user chose BST, I would read each line of the file given and store it into a tree and since the first file contains both numbers and words. I would be using this file to fill an empty tree that would traverse it at each index. Inside the loop I would make a variable which stores the values at the split of the line where there’s a space. After storing the values, I use an empty tree to store the values. To store it all, I would use a method, Insert, that takes the information at the starting index and an array that starts as the index after the first position and goes until it reaches an end. To store the number values, I declared them as a float to be able to insert them as an integer instead of a string.   
   For the hash table, I used the code provided with chaining and extra fields. To start, I would have to create a table of size 29. I then print the size of the table by getting the length of the hash. I then created a for loop to go through the big .txt file given to us. This time I separate the line like with the BST, I use line.index that separates the line where theres a space. To store the word value, I would have to store it in a variable that takes the first value of the file until it reaches the empty space. After that, I need to take the embedding and store both of the words and numbers of the file. This is done by using np.fromstring to get the strings of the file into the hash. The strings have to be embedded to start from the separator to the value -1 or until there are no more. I use the function dtype to declare the data type of the numbers(that were set to float) then I use the fumction sep that separates the values at an empty space. Now to create the hash table, I need to check if my load factor is 1. To do that I use the load factor method that returns the items divided by the length of the hash. If it is 1, I then would double the size of the hash and then insert the values of the old hash into the new ones, there would be a count that would increase by 1 each time I ran through that method. .In the if statement, I double the size of the method by first passing the hash table to it. In the double size method, I make the variable ‘new’ where I would double the size of the hash. To do that, I call the method where I create a hash table and pass 2 times the length of h plus 1 in order to double my table. I use 2 for loops for that where one lets me go through the table and the second lets me go through the buckets of the hash. Inside them I have an if statement that if the item in the has at the index is none, I then print nothing. Else, I call the insert method where I would pass the new variable which is 2 times the size of the original but have it be empty, then I pass the original hash with the index for the bucket, items, and 0 for the first position. I then pass the original hash again for the second values and so forth. Once the size is doubled, I return to making he hash table method and then I call the insert method where I pass the new variable that is doubled in size and the embedding from the file. The insert method would then insert the word and the embedding into the empty hash that is doubled in size. After that, the count increments. If the load factor is not 1 then I insert the word and embeddings into the hash table without doubling the hash. But I would also increment the count.
3. Compute and display statistics describing hash table- To start, I print the initial table size by getting the length of the hash table. I then run the code explained above, then I print the final table size again by getting the length of the hash. I then print the similarities then the load factor. To get the load factor I create a method that takes the hash and the number of items in it as parameters. TO display the percentage of empty lists, I use the round function to round the percentages and I then call the method numEmpty that gets the number of empty lists then I divide it by the length of the hash. To display the standard deviations, I use the statistics library and use statistics.stdev(lenList). lenList is used to find the length of the standard deviation, it appends all elements of the hash into a list. For the running time I use endTime and subtract it by startTime that gives me the length it took.
4. Read another file and find the similarities- For this method, I used a for loop to read the second text file and store its values in a variable. This time my list only has 2 words separated by comas. In this case I only need to access the text file and get the words from index one and zero. To get those words, I split the line by commas and save them onto their own variable that stores the first and second word (each respectively) Once they are stored, I appended them onto an empty list. I use variables e0 and e1 to find the similarities. E0 calls the method findHash which takes the hash and the data at index 0 as parameters. The same thing happens for e1. findHash is used to search the string in the tree and return the same number as if its in the hash. Afterwards, I return the words form that method to print the similarities between those words, I did that by getting the values of data from positions 0 to 2. To make the equation, I had to use mathematical and python logic to implement it correctly. I used the round and sum function while multiplying the values of e0 and e. Afterwards I had to divide the total, I used math.sqrt and applied it to the sum of e0^2. I used the sqrt function again to apply it to e1^2 while still using the sum function. To finish that, I pass the 4 into the round function that rounds the variable to the 4th decimal for the similarities. The Formula is as follows: “round(np.sum(e0 \* e1) / (math.sqrt(np.sum(e0 \* e0)) \* math.sqrt(np.sum(e1 \* e1)))”

**Experimental Results:**









**Conclusion:** I learned a lot on this lab. While learning to navigate and manipulate a hash table, I bettered my abilities to work with BST’s.

**Appendix:**

'''

Course: CS2302

Author: Erick Perchez

Assignment: Lab 2

Instructor: Dr. Fuentes

TA: Andita Nath

Date: 04/10/2019

Purpose: To use hash tables and binary trees to find similarities

between words using a long file given and my own file of

words

'''

import numpy as np

import time

import statistics

import math

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)

'''

===============================================================================

BST Section

'''

#def Building\_BST(file1,file2):

def findWord(T,k):

t = T

while t is not None:

if t.item[0] == k:

return t.item[1]

elif t.item[0] > k:

t= t.left

elif t.item[0]<k:

t = t.right

return None

def numNodes(T):

if T is None:

return 0

else:

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

return 0

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 Methods

'''

#finds number of items in table

def numItems(H):

Num=0

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

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

return Num

#counts how many are empty

def numEmpty(H):

count=0

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

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

count+=1

return count

#finds element in hash, returns -1 if not found

def findHash(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

#used to find the length of the standard diviation

def lenList(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):

new = 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(new,H.item[i][j][0],H.item[i][j][1])

return new

'''

===============================================================================

Main

'''

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

print('| Choose table implementation |')

print('| Type 1 for Binary Search Tree |')

print('| Type 2 for Hash Table with Chaining |')

print('|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|')

x = input('Choice: ')

print()

print('=====================================================')

print()

file1 = open('glove.6B.50d.txt',encoding='utf-8') #uses the text provided to be read later on

file2 = open('myFile.txt',encoding='utf-8') #uses my own text file to be read later on

if int(x) == 1:

print('Building Binary Search Tree...')

print()

T = None

start = int(time.time())

for line in file1:

info = line.split(' ')

#inserts words and embeddings text file

T = Insert(T, [info[0], np.array(info[1:]).astype(np.float)])

end = int(time.time())

print('Binary Search Tree Stats:')

print('Number of nodes:',numNodes(T))

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

print('Running Time for Binary Search Tree Construction:',(end-start))

print('Reading word file to determine similarities...')

print()

print('=====================================================')

print()

print('Word Similarities Found:')

start2=int(time.time())

for line2 in file2:

info = line2.split(',')

#returnslist when found

e0 = findWord(T,info[0])

e1 = findWord(T,info[1])

#Prints and compute the similarity

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

end2 = int(time.time())

print()

print('=====================================================')

print('Running Time for Binary Search Tree Query Processing:',(end2-start2))

elif int(x) == 2 :

print('Building Hash Table...')

print()

H = HashTableC(29)

print('Hash Table Stats:')

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

count=0

#used for big .TXT file given

for line in file1:

#gets index of first character in file

info = line.index(' ')

#gets first word

word = line[:info]

#creates embedding

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

if loadFactor(H,count) == 1:

#doubles size if loadfactor is 1

H=doubleSize(H)

InsertC(H,word,embedding)

count+=1

else:

InsertC(H,word,embedding)

count+=1

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

print('Reading word file to determine similarities...')

print()

print('=====================================================')

print()

print('Word Similarities Found:')

#used to store infro from file

l=list()

startTime=int(time.time())

#Used for my own .TXT file

for line in file2:

#gets index of first character in file

info = line.index(',')

#gets first word

word = line[:info]

#gets second word

word2=line[info+1:-1]

l.append([word,word2])

info=line.split(",")

#returns the list when found

e0=findHash(H, info[0])

e1=findHash(H, info[1])

#Prints and compute the similarity

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

endTime=int(time.time())

print()

print('=====================================================')

print()

print('Load Factor:', loadFactor(H, numItems(H)))

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

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

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

print()

print('=====================================================')

else:

print ('Invalid Input')

file1.close()

file2.close()

“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”