Introduction:

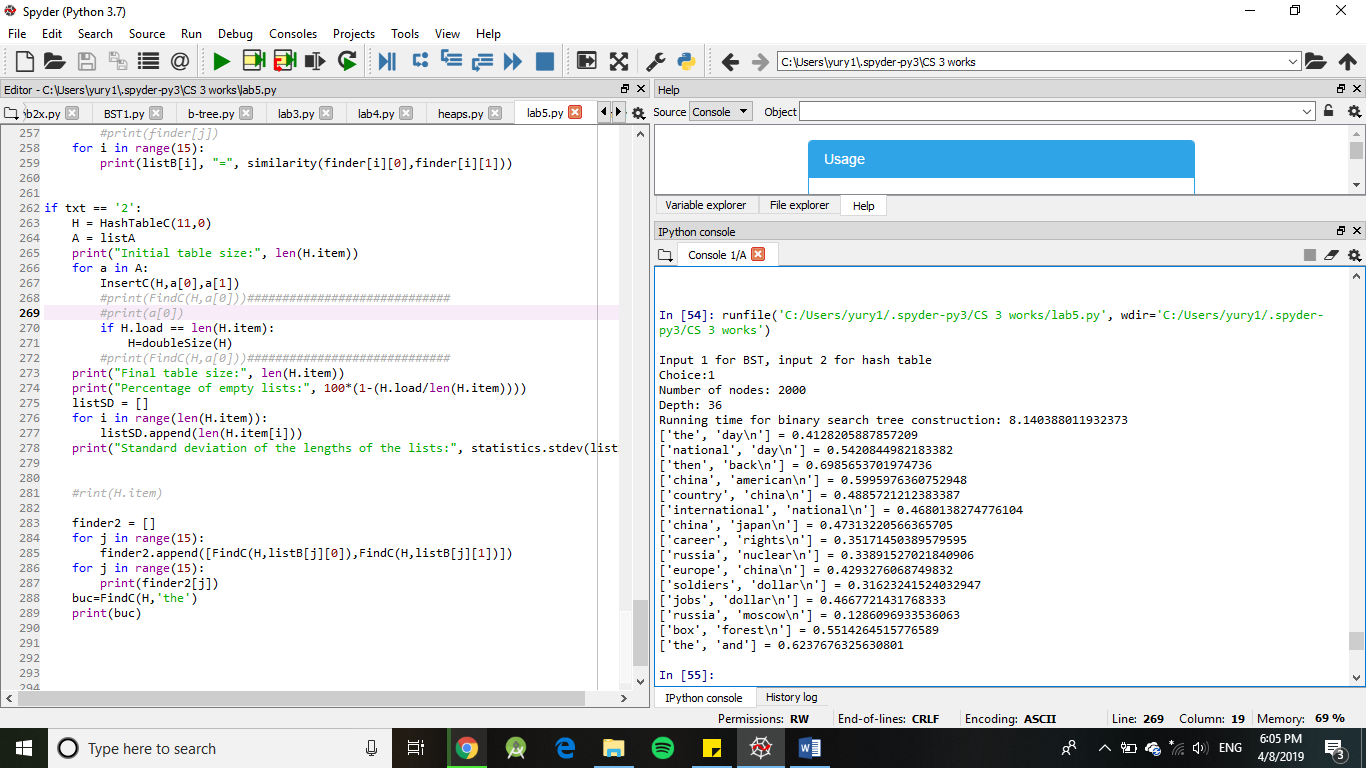
The problem that was to be solved was to read a file and store into a bst and hash table

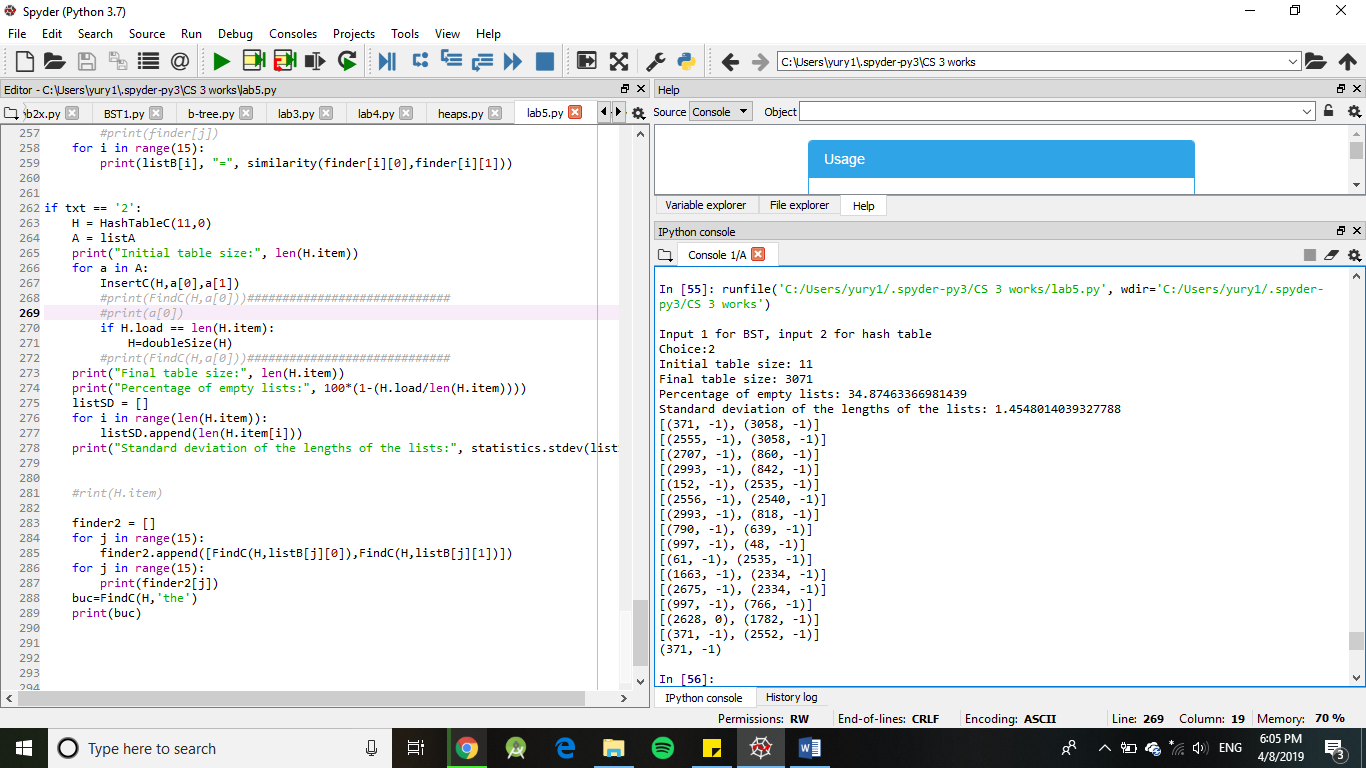
Solution:

I first attempted at the problem by writing pseudocode and trying to make sense of it. I then implemented it into code and ran several times. Each time I ran it I got close to solving the problem. I finally followed it up with actually coming up with the solution.

Results:

The experiments included running the program several times, and finding out the outputs. The running times.





Conclusions:

I learned that finding similarities of a hash table is very difficult.

I Yury Ionov 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.



Source code:

# Code to implement hash tables and bsts

# Programmed by Yury Ionov

# Instructor: Olac Fuentas

# TA: Mali, Anindita Nath

# LAB 5

# Last modified March 27, 2019

import time

#import matplotlib.pyplot as plt

import numpy as np

import math

import statistics

class BST(object):

#contstructor

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

self.item = item

self.left = left

self.right = right

s = item[0]

t = [ord(c) for c in s]

total = 0

for i in range (len(t)):

total = total + t[i]

self.total = total

def getDepth(self):

if self.left and self.right:

return 1 + max(self.left.getDepth(), self.right.getDepth())

elif self.left:

return 1 + self.left.getDepth()

elif self.right:

return 1 + self.right.getDepth()

else:

return 1

start\_time = time.time()

def Insert(T, newItem):

charNums = []

for i in range(len(newItem[0])): #turns the word to ascii number

charNums.append(ord(newItem[0][i]))

total = sum(charNums)

if T == None:

T = BST(newItem)

elif total<T.total:

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)

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])

InOrderD(T.left,space+' ')

def numNodes(T):

if T is None:

return 0

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

def Depth(T):

if T is None:

return 0 ;

else :

# Compute the depth of each subtree

lDepth = Depth(T.left)

rDepth = Depth(T.right)

# Use the larger one

if (lDepth > rDepth):

return lDepth+1

else:

return rDepth+1

def FindBST(T,k):

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

s = k

t = [ord(c) for c in s]

total = 0

for i in range (len(t)):

total = total + t[i]

if T is None or T.total == total:

return T

if total>T.total:

return FindBST(T.right,k)

return FindBST(T.left,k)

class HashTableC(object):

# Builds a hash table of size 'size'

# Item is a list of (initially empty) lists

# Constructor

def \_\_init\_\_(self,size,load):

self.item = []

for i in range(size):

self.item.append([])

self.load=0

def InsertH(H,k):

# Inserts k in appropriate bucket (list)

# Does nothing if k is already in the table

charNums = []

for i in range(len(k[0])): #turns the word to ascii number

charNums.append(ord(k[0][i]))

total = sum(charNums)

b = total%len(H.item)

H.item[b].append(k)

H.load = H.load +1

def FindH(H,k):

# Returns bucket (b) and index (i)

# If k is not in table, i == -1

charNums = []

for i in range(len(k[0])): #turns the word to ascii number

charNums.append(ord(k[0][i]))

total = sum(charNums)

b = total%len(H.item)

try:

j = H.item[b].index(total)

except:

j = -1

return b, j

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])

H.load = H.load +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 b, i

return b, -1

def h(s,n):

r = 0

for c in s:

r = (r\*7 + ord(c))% n

return r

def DeleteC(H,k):

# Returns k from appropriate list

# Does nothing if k is not in the table

# Returns 1 in case of a successful deletion, -1 otherwise

b = k%len(H.item)

try:

H.item[b].remove(k)

return 1

except:

return -1

def doubleSize(H):

H2 = HashTableC(len(H.item)\*2+1, 0)

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

for i in H.item[b]:

InsertH(H2,i)

return H2

def similarity(a,b):

array1 = a.item[1]

array2 = b.item[1]

dotProduct = 0

for i in range(50):

dotProduct = dotProduct + (array1[i]\*array2[i]) # gets the dot product

magnitudeOfA = 0

for i in range(50):

magnitudeOfA = magnitudeOfA +(array1[i]\*array1[i])

magnitudeOfA = math.sqrt(magnitudeOfA)

magnitudeOfB = 0

for i in range(50):

magnitudeOfB = magnitudeOfB +(array2[i]\*array2[i])

magnitudeOfB = math.sqrt(magnitudeOfB)

return dotProduct/(magnitudeOfA\*magnitudeOfB)

#a.numpy()

def numOfFileLines(fileName):

count = 0

with open(fileName, 'rb') as f:

for line in f:

count +=1

return count

txt = input("Input 1 for BST, input 2 for hash table\nChoice:")

File = open('C:/Users/yury1/Desktop/data.txt', encoding='utf-8')

#contents = File.read()

lines = File.readlines()

#for i in range(5):

#print(lines[i])

listA = []

numOfLines = numOfFileLines('C:/Users/yury1/Desktop/data.txt')

for j in range(2000):

a = lines[j].split(" ", 1) #splits only at the first space

a[1]=a[1].split(" ")

for i in range(50):

a[1][i] = float(a[1][i])

listA.append([a[0],np.array(a[1])]) #appends the word and the embedding

"""

for i in range(5):

print(listA[i])

"""

File2 = open('C:/Users/yury1/Desktop/data2.txt', encoding='utf-8')

lines2 = File2.readlines()

listB = []

#for j in range(15):

for line in lines2:

a = line.split(",")

listB.append([a[0],a[1]])

#

#for j in range(15):

# print(listB[j])

if txt == '1':

T =None

for a in range(2000):

T = Insert(T,listA[a])

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

print("Depth:", Depth(T))

print("Running time for binary search tree construction:",time.time() - start\_time)

#InOrderD(T, " ")

"""

print(listB[1][1])

print(FindBST(T,"the"))

#InOrder(T)

"""

finder = []

for j in range(15):

finder.append([FindBST(T,listB[j][0]),FindBST(T,listB[j][1])])

#for j in range(15):

#print(finder[j])

for i in range(15):

print(listB[i], "=", similarity(finder[i][0],finder[i][1]))

if txt == '2':

H = HashTableC(11,0)

A = listA

print("Initial table size:", len(H.item))

for a in A:

InsertC(H,a[0],a[1])

#print(FindC(H,a[0]))#############################

#print(a[0])

if H.load == len(H.item):

H=doubleSize(H)

#print(FindC(H,a[0]))#############################

print("Final table size:", len(H.item))

print("Percentage of empty lists:", 100\*(1-(H.load/len(H.item))))

listSD = []

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

listSD.append(len(H.item[i]))

print("Standard deviation of the lengths of the lists:", statistics.stdev(listSD))

#rint(H.item)

finder2 = []

for j in range(15):

finder2.append([FindC(H,listB[j][0]),FindC(H,listB[j][1])])

for j in range(15):

print(finder2[j])

buc=FindC(H,'the')

print(buc)