Laila Romero

CS 2302 Data Structures

April 9, 2019

Lab 5

* Introduction

For this lab we were trying to read in a file and store the contents into either a binary tree or hash table with chaining depending on the user input. We are also to increase the table size every time the load factor reaches 1. After this we are to print the statistics for the BST and hash table such as nodes, height, and running time. Then we are to read in another file containing a variation of words that would be compared to each other and print out the similarities between the words on a scale of 1 to -1, 1 being the closest in similarity and -1 being the furthest.

• Proposed Solution Design and Implementation

To read in the glove file I open the file and read it in line by line to create an array. I also split the file by spaces so that the numbers could print individually and then split it again by excluding the first element since it was a word and not a number. I then insert it to the hashtable using the Insert function.

To read in the comparisons I read it in line by line also and split the file by spaces. I then used the method I named similarity to print the cosine similarity of the two words in each line of the file. The similarity function took the array of words and looked for the two words in every line. It then initialized the variables called top, bottom\_a, and bottom\_b to a value of 0. It then entered a for loop traversing through the length of the first word and increases top to the float values of word1 and word2. Bottom\_a increases to the float value of word1\* 2 while bottom\_b increases to the float value of word2\*2. After exiting the for loop, the bottom\_a value is then equal to the square root of bottom\_a and bottom\_b is equal to the square root of bottom\_b. After this it finally returns the top value divided by bottom\_a \* bottom\_b.

To calculate running time I used start = time.time once the user made a selection between the hash table and the binary tree and then used end = time.time once the similarities were finished. And to calculate the total running time I subtracted the start from the end.

For doubling the size of the hashtable, I made a doubleSize method. It takes the length of the original hashtable and multiples it by 2 and adds 1. It then reinserts the items into the new hashtable and returns the new hashtable.

For calculating percentage of empty lists, I start with a variable j with an initial value of 0. It then enters a for loop traversing the length of the hashtable. It then checks if the length of the item in the hashtable is 0 and if it is it changes the value of j to j+1. After exiting the for loop the method then returns j/length of item in hashtable \* 100 to get percentage.

• Experimental results

In order to test my program, I printed out the first 50 lines of the glove fila to see if it printed properly in the hashtable and the binary search tree. I then tried testing to see if the words were separated from the numbers for both implementations. I had a bit of trouble getting the binary search tree to insert and print properly but was able to get the hashtable to print just fine. The running time for both the methods was around 0.004280805587768555. Below I have stated the big O time complexities of the methods in the program.

The similarity method used to compare the two words in the lines of the comparisons file had a big O complexity of O(nlogn).

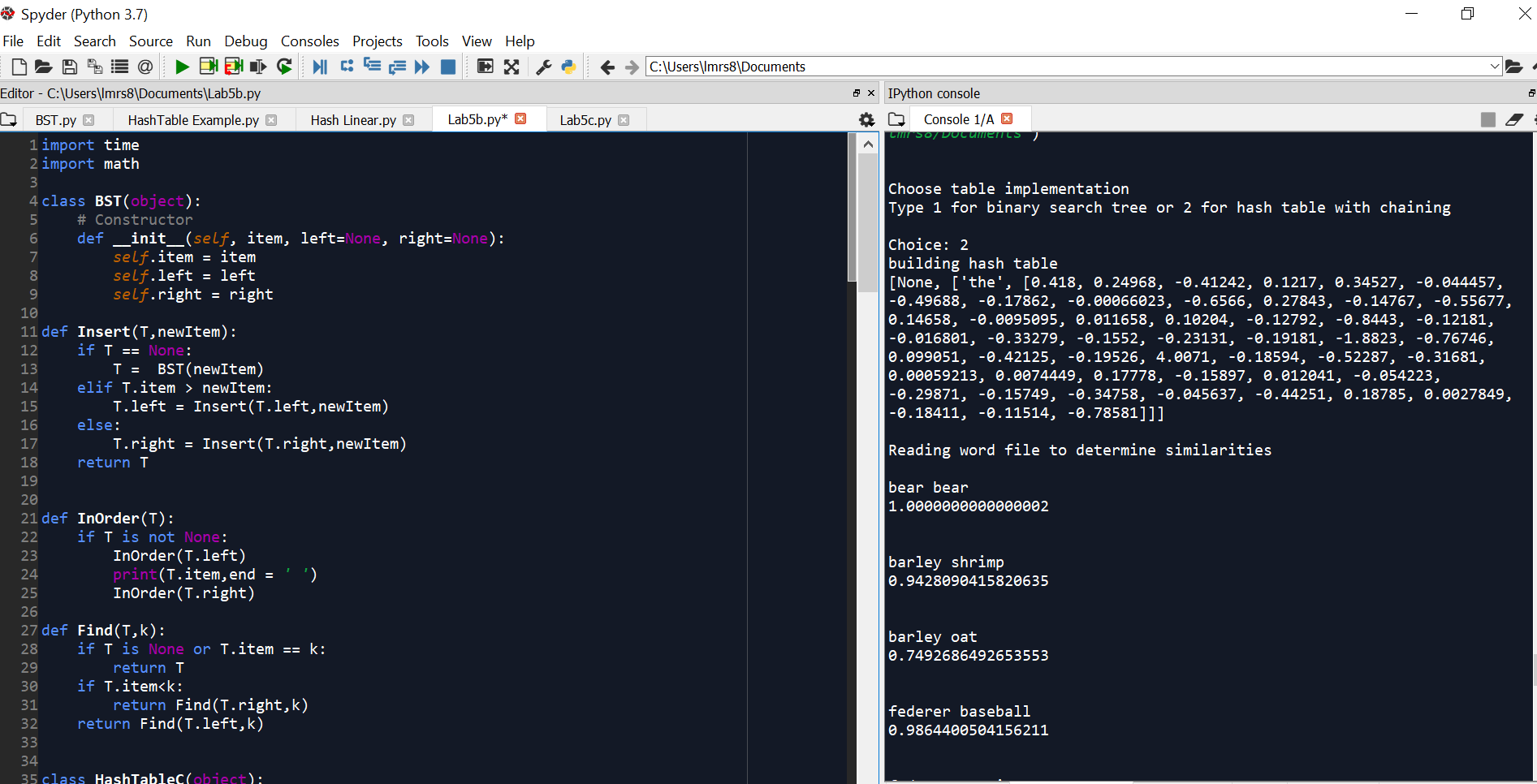
The doubling size method used to double the size of the hashtable after the load factor reached 1 had a big O complexity of O(n2).

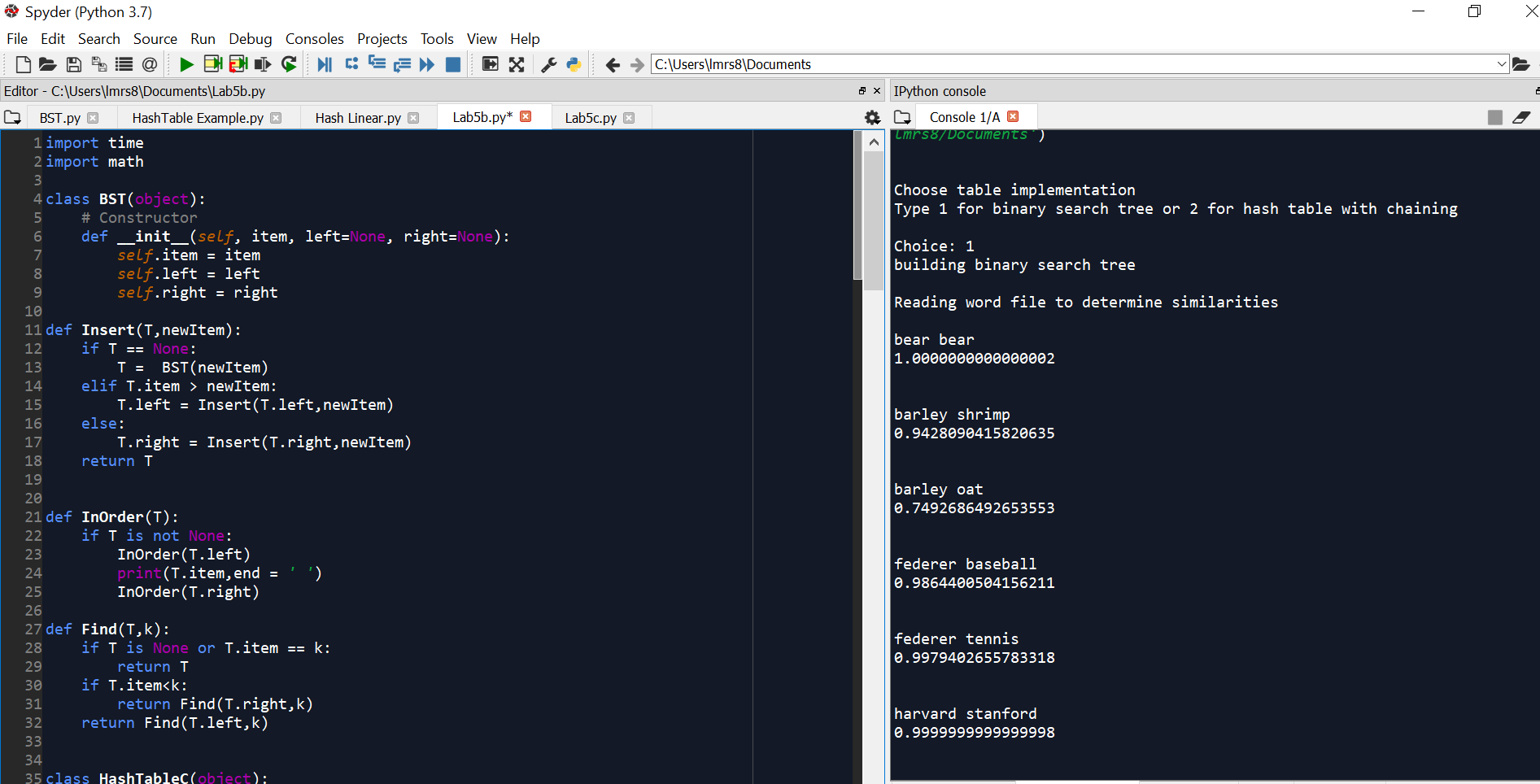
The Recompute method used for recomputing the hashtable after reading in the file had a big O complexity of O(n2).

The percentage method used for calculating the percentage of empty lists in the hashtable had a big O complexity of O(n).

The spaces2 method which split the file by spaces and inserted them into an array was O(n2)

|  |  |
| --- | --- |
| Method | Big O |
| Similarity method | O(nlogn) |
| DoubleSize method | O(n2) |
| ReCompute method | O(n2) |
| Percentage method | O(n) |
| Spaces2 | O(n2) |





• Conclusion

In conclusion, I learned how to read in files and split and inset them into arrays. I then learned how to insert them in to both hashtables and binary search trees. I also learned how to implement the cosine similarity to the file of words and show how closely related or similar the words are.

Although my method of binary search tree did not work, I understood the concept behind the method and just needed a little more to properly implement it.

• Appendix – Source codes

import time

import math

import numpy as np

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

def InOrder(T):

if T is not None:

InOrder(T.left)

print(T.item,end = ' ')

InOrder(T.right)

def Find(T,k):

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

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

class HashTableC(object):

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

self.item = []

for i in range(size):

self.item.append([])

def InsertC(H,k,l):

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

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

def FindC(H,k):

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\*n + ord(c))% n

return r

def similarity(H,w1,w2):

ww1 = FindC(H,w1)

ww2=FindC(H,w2)

top = 0

bottom\_a=0

bottom\_b=0

for i in range(len(ww1)):

top += float(ww1[i])\*float(ww2[i])

bottom\_a += float(ww1[i])\*\*2

bottom\_b += float(ww2[i])\*\*2

bottom\_a=math.sqrt(bottom\_a)

bottom\_b=math.sqrt(bottom\_b)

return top/(bottom\_a\*bottom\_b)

def Recompute(H):

temp = HashTableC(len(H.item))

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

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

InsertC(temp,H.item[i][j][0], H.item[i][j][1])

return temp

def LargeList(numLines, text):

newFile= open(text, encoding='utf8')

newList= [None]\*numLines

for i in range(numLines):

newList[i]=newFile.readline()

return newList

print('', end='\n' )

def ListMaker(theList):

newList=[]

for i in range (len(theList)):

innerList = theList[i].split(" ",1)

innerListNums = innerList[0].split(" ")

for j in range (len(innerListNums)):

innerListNums[j]= float(innerListNums[j])

newList.append([innerList[1], innerListNums])

return newList

def spaces2(fileName):

newList=[None]

for i in range(1):

innerList = fileName.split(" ",1)

innerListNums = innerList[1].split(" ")

for j in range (len(innerListNums)):

innerListNums[j]= float(innerListNums[j])

newList.append([innerList[0], innerListNums])

return newList

def doubleSize(H):

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

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

for i in H.item[b]:

InsertC(H2,i)

return H2

def Percentage(H):

j=0

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

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

j=j+1

return j/len(H.item)\*100

def loadFactor(H):

return np.sum(H.item>=0)/len(H.item)

path = 'C:/Users/lmrs8/Documents/glove.6B.50d.txt'

text\_file=open(path, encoding= "utf-8")

text\_file=open(path, 'r')

text\_file=open(path, errors='ignore')

text=text\_file.readline()

T = None

A = spaces2(text)

for a in A:

# T = Insert(T,a)

print(end='\n')

InOrder(T)

file = 'glove.6B.50d.txt'

size = 11

H = HashTableC(size)

print("Choose table implementation")

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

choice = int(input("Choice: "))

if choice ==1:

start = time.time()

print("building binary search tree")

Hr= Recompute(H)

print(end='\n')

print('Reading word file to determine similarities')

with open("comparisons.txt") as compares:

for line in compares:

string2 = line.split()

print(end='\n')

print(string2[0]+" "+string2[1]+ " ")

print(str(similarity(Hr, string2[0], string2[1])))

print(end='\n')

end=time.time()

print('Running Time', end-start)

if choice ==2:

start = time.time()

print("building hash table")

print(spaces2(text))

Hr= Recompute(H)

print(end='\n')

print('Initial size ', size)

print('New Size ', len(H.item)\*2+1)

# print('Load Factor ', loadFactor(H))

# print(Percentage(H))

print(end='\n')

print('Reading word file to determine similarities')

with open("comparisons.txt") as compares:

for line in compares:

string2 = line.split()

print(end='\n')

print(string2[0]+" "+string2[1]+ " ")

print(str(similarity(Hr, string2[0], string2[1])))

print(end='\n')

end=time.time()

print('Running Time', end-start)

“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.”

