LAB 5

CS2302-DATA STRUCTURES

Quinones Rodriguez, Diego A
80582918

Report

Lab 5 required use to use BST and Hash Table with chaining in simple words with the purpose of comparing two words in a .txt file and getting the percentage of difference between them, among another info details like:

Number of nodes, height, running times, initial table size and final table size.

The code starts asking the user for an input between 1 and 2, 1 being the binary search tree and 2 being the hash table, I used an if to access each of the classes accordingly.

BST starts by using a method that creates the tree with the glove.6B file and returns the tree, after that it uses a method that determines how similar a word is to the next one, and it uses the words.txt.

HashTable starts by using a method that creates the table with the glove.6B file and returns the table, after that it uses a method that determines how similar a word is to the next one, and it uses the words.txt.

Screenshots

```
In [170]: runfile('/Users/diegoquinones/Desktop/CS Data Structures/
Lab5.py', wdir='/Users/diegoquinones/Desktop/CS Data Structures')

Choose table implementation Type 1 for binary search tree or 2 for hash
table with chaining2
Choice:  2
Building hash table with chaining Initial Size:  23
Final Table Size:  23
Percentage of empty lists:  0.0  %
Standard deviation of the lengths of the lists:  1.3232514177693764
Reading word file to determine similarities

Word similarities found:

Similarity ['bear', 'bear '] = 0.46861731182541944
Similarity ['barley', 'shrimp '] = -1.8673839675110195
Similarity ['barley', 'oat '] = -1.8673839675110195
Similarity ['federer', 'baseball '] = -0.34394303721576414
Similarity ['federer', 'tennis '] = -0.34394303721576414
Similarity ['harvard', 'stanford '] = 1.1657917451376438
Similarity ['harvard', 'utep '] = 1.1657917451376438
Similarity ['harvard', 'ant  '] = 1.1657917451376438
Similarity ['raven', 'crow  '] = 1.9562776279569056
Similarity ['raven', 'whale  '] = 1.9562776279569056
Similarity ['spain', 'france  '] = 0.5466274878909728
Similarity ['spain', 'mexico  '] = 0.5466274878909728
Similarity ['mexico', 'france '] = 1.1311418353281295
Similarity ['mexico', 'guatemala  '] = 1.1311418353281295
Similarity ['computer', 'platypus'] = -0.13488927298110215
16.607399940490723
```

```
Choose table implementation Type 1 for binary search tree or 2 for hash
table with chaining2
Choice:  2
Building hash table with chaining Initial Size:  23
Final Table Size:  23
Percentage of empty lists:  0.0  %
Standard deviation of the lengths of the lists:  1.3232514177693764
Reading word file to determine similarities

Word similarities found:

Similarity ['bear', 'bear '] = 0.46861731182541944
Similarity ['barley', 'shrimp '] = -1.8673839675110195
Similarity ['barley', 'oat '] = -1.8673839675110195
Similarity ['federer', 'baseball '] = -0.34394303721576414
Similarity ['federer', 'tennis '] = -0.34394303721576414
Similarity ['harvard', 'stanford '] = 1.1657917451376438
Similarity ['harvard', 'utep '] = 1.1657917451376438
Similarity ['harvard', 'ant  '] = 1.1657917451376438
Similarity ['raven', 'crow  '] = 1.9562776279569056
Similarity ['raven', 'whale  '] = 1.9562776279569056
Similarity ['spain', 'france  '] = 0.5466274878909728
Similarity ['spain', 'mexico  '] = 0.5466274878909728
Similarity ['mexico', 'france '] = 1.1311418353281295
Similarity ['mexico', 'guatemala  '] = 1.1311418353281295
Similarity ['computer', 'platypus'] = -0.13488927298110215
16.607399940490723

In [171]: runfile('/Users/diegoquinones/Desktop/CS Data Structures/
Lab5.py', wdir='/Users/diegoquinones/Desktop/CS Data Structures')
```

```
In [171]: runfile('/Users/diegoquinones/Desktop/CS Data Structures/
Lab5.py', wdir='/Users/diegoquinones/Desktop/CS Data Structures')

Choose table implementation Type 1 for binary search tree or 2 for hash
table with chaining1
Choice:  1
Building binary search tree
Number of nodes:  400000
Reading word file to determine similarities

Word similarities found:

Similarity ['bear', 'bear '] = 0.8736806978475604
Similarity ['barley', 'shrimp '] = 0.532376436699998
Similarity ['barley', 'oat '] = 0.7439777972790234
Similarity ['federer', 'baseball '] = 0.3621468102010926
Similarity ['federer', 'tennis '] = 1.0098835498532472
Similarity ['harvard', 'stanford '] = 0.3621468102010926
Similarity ['harvard', 'utep '] = 0.5941882523380748
Similarity ['harvard', 'ant  '] = 0.5941882523380748
Similarity ['raven', 'crow  '] = 0.6976618696275732
Similarity ['raven', 'whale  '] = 0.5941882523380748
Similarity ['spain', 'france  '] = 0.6718875809588796
Similarity ['spain', 'mexico  '] = 0.6718875809588796
Similarity ['mexico', 'france '] = 0.532376436699998
Similarity ['mexico', 'guatemala  '] = 0.4779606535932242
Similarity ['computer', 'platypus'] = 1.0085489883957603
14.200464963912964

In [172]:
```

```
Choose table implementation Type 1 for binary search tree or 2 for hash
table with chaining2
Choice:  2
Building hash table with chaining Initial Size:  23
Final Table Size:  23
Percentage of empty lists:  0.0  %
Standard deviation of the lengths of the lists:  1.3232514177693764
Reading word file to determine similarities

Word similarities found:

Similarity ['bear', 'bear '] = -1.0
Similarity ['barley', 'shrimp '] = -1.0
Similarity ['barley', 'oat '] = -1.0
Similarity ['federer', 'baseball '] = -1.0
Similarity ['federer', 'tennis '] = -1.0
Similarity ['harvard', 'stanford '] = -1.0
Similarity ['harvard', 'utep '] = -1.0
Similarity ['harvard', 'ant  '] = -1.0
Similarity ['raven', 'crow  '] = -1.0
Similarity ['raven', 'whale  '] = -1.0
Similarity ['spain', 'france  '] = -1.0
Similarity ['spain', 'mexico  '] = -1.0
Similarity ['mexico', 'france '] = -1.0
Similarity ['mexico', 'guatemala  '] = -1.0
Similarity ['computer', 'platypus'] = 1.0
13.295428037643433
```

Running Times

Hash Table:

16.60

15.1787

13.0164

Binary Search:

14.2

13.0218

11.4848

Source Code

```
#!/usr/bin/e
nv python3
                # -*- coding: utf-8 -*-
                """
                Created on Mon Apr  1 21:50:50 2019
                @author: diegoquinones
                """


                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

                # Implementation of hash tables with chaining using strings
                #All of this code belongs to hash table
```

```
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([])
        self.num_items=0

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 EmptyPer(H):
    counter=0
    for i in H.item:
        if i ==[]:
            counter=+1
    return counter


def Similarity(H,file):
    print('Reading word file to determine similarities ')
```

```
    print()
    print('Word similarities found:')
    print()
    for i in file:
        word=i.split(',')
        word[1]=word[1].replace('\n',' ')
        a=FindC(H,word[0])
        b=FindC(H,word[1])
        print('Similarity',
word,'=',round(np.sum(a[2]*b[2]))/((math.sqrt(np.sum(a[2]*a[2])))*(math.sqrt(np.sum(b[2
]*b[2])))))


def HashBuilder(v):
    print('Building hash table with chaining',end=' ')
    H1=HashTableC(23)
    print('Initial Size: ', len(H1.item))
    for i in v:
        line = i
        letters=line.split(" ")
        word=letters[0]
        embed = np.empty([50], dtype=float)
        counter=0
        for j in range(1,len(letters)):
            embed[counter]=letters[j]
            counter+=1
        InsertC(H1,word,embed)
        H1.num_items+=1
    EmptyLists=((EmptyPer(H1))/len(H1.item) )*100
    deviation=SDev(H1)
    print('Final Table Size: ',len(H1.item))
    print('Percentage of empty lists: ',EmptyLists,' %')
    print('Standard deviation of the lengths of the lists: ',deviation)
    return H1

#doubles size of hash table
def ExpandHash(H):
    #creates longer hash
    H1= HashTableC((len(H.item)*2)+1)

    #inserts values to new hash
    for i in range(len(H.item)):
        for j in H.item[i]:
```

```
            InsertC(H1,j[0],j[1])
    return H1



def loadFactor(H):
    return H.num_items//len(H.item)




def SDev(H):
    a=0
    k=loadFactor(H)
    for i in H.item:
        a=a+ len(i)-k
    standard=((1/len(H.item)*a)/(len(H.item)))*100
    return standard

#All of this code belong to BST


def Tree(a):
    print('Building binary search tree')
    nodeCounter=0
    tree=None
    for i in a:
        line=i
        text=line.split(" ")
        word = text[0]
        j= value(word)
        count=0
        e=np.empty([50],dtype=float)
        for k in range(1,len(text)):
            e[count]=text[k]
            count+=1
        tree=Insert(tree,[j,word,e])
        nodeCounter+=1
    print('Number of nodes: ',nodeCounter)
    return tree
```

```
def Insert(T,newItem):
    if T==None:
        T=BST(newItem)
    elif T.item[0]>newItem[0]:
        T.left=Insert(T.left,newItem)
    elif T.item[0]<newItem[0]:
        T.right=Insert(T.right,newItem)
    return T


def height(T):
    counter=0
    temp1=T
    while temp1 is not None:
        counter=counter+1
        temp1=temp1.left
    counter=0
    return temp1


def value(w):
    num=[ord(c) for c in w]
    counter=0
    for i in num:
        counter=counter+1
    return counter


def wordfinder(T,k):
    while T is not None:
        if T.item[0]==value(k) or T.item[1]==k:
            return T.item
        elif T.item[0]<value(k):
            T=T.right
        else:
            T=T.left
    return None


def similaritybst(T,file):
    temp=T
    print('Reading word file to determine similarities')
```

```
    print()
    print('Word similarities found: ')
    print()
    for words in file:
        words=words.split(',')
        words[1]=words[1].replace('\n',' ')
        b=wordfinder(temp,words[1])
        a=wordfinder(temp,words[0])
        print('Similarity',
words,'=',round(np.sum(a[2]*b[2]))/((math.sqrt(np.sum(a[2]*a[2])))*(math.sqrt(np.sum(b[
2]*b[2])))))


#main method
userinput= input('Choose table implementation Type 1 for binary search tree or 2 for
hash table with chaining')
file=open("glove.6B.50d.txt","r")
words=open("words.txt","r")
type(userinput)
if userinput=='2':
    start = time. time()
    print('Choice: ',2)
    HT=HashBuilder(file)
    Similarity(HT,words)
    end = time. time()
    print(end - start)

if userinput=='1':
    start = time. time()
    print('Choice: ',1)
    BS=Tree(file)
    similaritybst(BS,words)
    end = time. time()
    print(end - start)
```

Academic Honesty Certification

“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.”

-Diego Quinones