CS3 Lab 4 Report

Daniela Flores

Intro

In this lab assignment I had to read a file which contained a large amount of words and their embeddings. We had to give the user the choice of choosing either a hash table or a BST to store the data in the file. After the user decides which data structure they want, the data structure is then built, after its done storing the words/embeddings into the data structure, information about the data structure is displayed. Another file then is read which consists of pairs of words. The similarity of these two words is then computed by a formula provided to us by Dr.Fuentes. After the similarity of the words is done computing, the runtime is displayed.

Proposed Solution and Implementation:

I wrote the method to store the data in the text file into a hash table first by initializing a variable into a hash table of size 11. A for loop then iterates through the lines of the text file and splits the elements of every line. Then I stored the first element of the line, which is the word, into a variable and then the rest of its embedding into another variable. I then used a for loop to traverse through the embeddings and turn the strings into floats. After that is done the word and the embedding is added to the hash table. After the hash table is done, the initial size of the hash table, the final size of the hash table, the load factor, and the percentage of empty lists in the hash table is displayed.

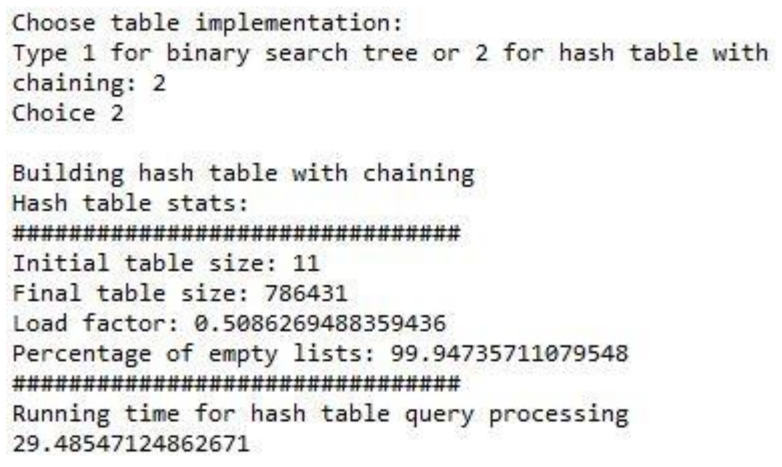
The method to store the elements in the text file into a BST is first started by initializing the variable that'll store the BST, afterwards, the file is then read by a for loop. Inside the for loop the line of a file is split. The way the data is stored into the BST is the same as the hash table, that is, the first word Is stored in a variable and the embedding is turned from strings into floats. After the BST is done, the number of nodes and the height of BST is displayed.

The method to compare two words is first started by splitting the words of the text file and storing each of the words into different variables. An if statement checks if the user chose a hash table or a BST and calls the corresponding method to compute their similarity with the function given to us by Dr.Fuentes. This formula to compute the similarity of two words consists of dividing the dot product by the magnitudes of their embeddings.

After I was done writing all the necessary methods, I started working on what would be first displayed when the program was ran. When the program is first running it asks the user to type one BST or two for hash table for the table implementation. Two if statements are used to call either the method for BST or for a hash table, depending on the user’s input. If another number other than 1 or 2 is typed an error message is displayed.

Experimental Results:

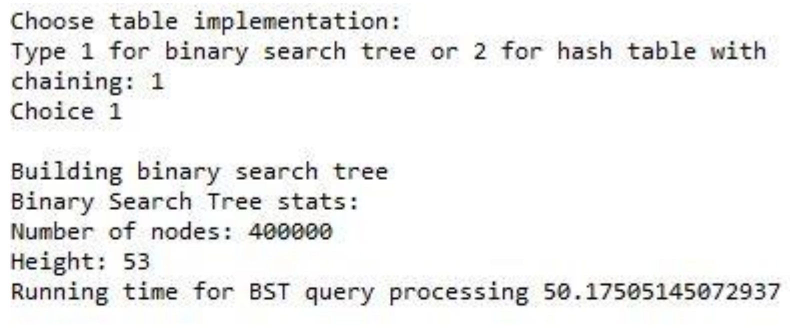
I experimented a lot with my method to find the similarity of two words by changing how to calculate the dot product and the magnitude as I kept getting errors. I didn’t end up fully resolving this issue. That is why in the screenshots below, the similarity of two words file is not shown. I effectively was able to scan the file and store the two words into variables, I think where I went wrong was finding the embedding of the words. The method is still in my program, I just never call it as my program didn't compile correctly if I called it. Below the output of my program is shown.



```
Choose table implementation:
Type 1 for binary search tree or 2 for hash table with
chaining: 2
Choice 2

Building hash table with chaining
Hash table stats:
################################
Initial table size: 11
Final table size: 786431
Load factor: 0.5086269488359436
Percentage of empty lists: 99.94735711079548
################################
Running time for hash table query processing
29.48547124862671
```

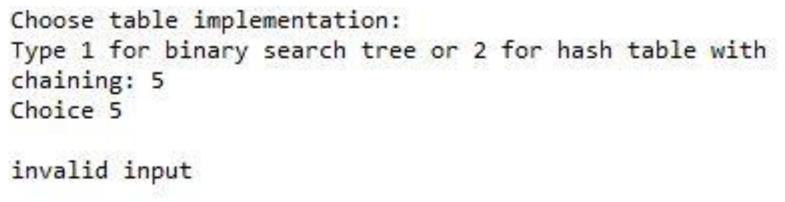
****



```
Choose table implementation:
Type 1 for binary search tree or 2 for hash table with
chaining: 1
Choice 1

Building binary search tree
Binary Search Tree stats:
Number of nodes: 400000
Height: 53
Running time for BST query processing 50.17505145072937
```

****



```
Choose table implementation:
Type 1 for binary search tree or 2 for hash table with
chaining: 5
Choice 5

invalid input
```

Conclusion:

What I learned from this lab was how to work with a hash table and how to compute the stats of that hash table like its percentage of empty lists. I also got to learn how to read a file and split its contents. I also got more practice with getting input from the user.

Appendix:

#Course:CS2302

#aAuthor:Daniela Flores

#Lab5

```
#Instructor:Olac Fuentes

#T.A: Dita Nath

#date of last Mod: 4/2/19

#purpose of program: in this program I had to scan a file and store its contents in a hash table or a BST
whatever the user

#desires,various functions are then performed to the data structure, to finalize, a function to find the

#similarity of a set of words is then performed given either a hash table or a BST

import math

import time


class HashTableC(object):

    def __init__(self,size):

        self.item = []

        for i in range(size):

            self.item.append([])

        self.num_Items = 0


def InsertC(H,k,l):

    H.num_Items = H.num_Items+1

    #Inserts k in appropriate bucket(list)

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

    if H.num_Items / len(H.item) >= 1.0:

        #increments capacity of hash table

        for i in range(len(H.item)+1):

            H.item.append([])

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

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

```
def FindC(H,k):
    #Returns bucket and index of given element 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
#computes load factor
def loadFactor(H):
    if H is None:
        return None
    counter = 0
    #counts the number of items in hashtable
    for i in range(len(H.item)):
        counter =counter + len(H.item[i])
    #returns computed load factor
    return counter/len(H.item)
#computes the percenatge of empty lists
def emptyListsPerc(H):
    if H is None:
        return None
    #checks number of empty lists we have in our hash table
    emptyLists = 0
    for i in range(len(H.item)):
        if len(H.item[i]) == 0:
            emptyLists = emptyLists+ 1
    #computes percentage
    percentage = (emptyLists * 100) / len(H.item)
    return percentage
#starts hash table of size 11
H = HashTableC(11)

##################################################
H = HashTableC(11)
#method that builds the hash table and populates it with the contents in the file
def hashTableBuilder():
    f = open('glove.6B.50d.txt',encoding='utf-8')
    originalSize = len(H.item)
    #will store the embedding
    embedding = []
    #traverses through the lines of the file
    for line in f:
        temp = line.split(" ")
        word = temp[0]
        emb = temp[1:]
        #turns string into floats
        for i in range(len(emb)):
            embedding.append(float(emb[i]))
        #inserts word and embedding into hash table
        InsertC(H,word, embedding)
    print('################################')
    print('Initial table size:', originalSize)
    print('Final table size:', len(H.item))
    print('Load factor:', loadFactor(H))
    print('Percentage of empty lists:', emptyListsPerc(H))
    print('################################')
##################################################
class BST(object):
    # Constructor
    def __init__(self, item, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right
#inserts new item into BST
def Insert(T,newItem):
    if T == None:
        T = BST(newItem)
    elif T.item[0] > newItem[0]:
        T.left = Insert(T.left,newItem)
    else:
        T.right = Insert(T.right,newItem)
    return T
#computes the number of nodes in BST
def countNodes(T):
    count = 0
    if T is not None:
        count = count+ 1
    else:
        return 0
    return count + countNodes(T.right) + countNodes(T.left)
#gets height of BST by traversing through it till it reaches the leafs
```

```
def height(T):
    if T is None:
        return 0
    leftB = 1+height(T.left)
    rightB = 1+height(T.right)
    #checks which height of left and right subtree is larger and returns it
    if leftB > rightB:
        return leftB
    return rightB
#finds item in BST
def Find(T,k):
    while T is not None:
        if T.item[0] == k:
            return T.item
        elif T.item[0] < k:
            T = T.right
        else:
            T = T.left
    return -1
#builds BST with the contents of the txt file
def BSTBuilder():


    T = None
    f = open('glove.6B.50d.txt',encoding='utf-8')
    for line in f:
        embedding = []
        temp = line.split()
        word  = temp[0]
        emb = temp[1:]
        #turns string to float
        for i in range(len(emb)):
            embedding.append(float(emb[i]))
        T = Insert(T,[word,embedding])
    print('Number of nodes:',countNodes(T))
    print('Height:', height(T))
    #comparingTextFile(T,1)



##################################################
#method that compares two words of a file through the words embedding
def comparingTextFile(H,input):
    f = open('similarityFile.txt')
    for line in f:
        temp = line.split(" ")
        #stores first word
        one = temp[0]
        #stores second word
        two = temp[1]
        #if user chose hash table
        if input == 2:
            print('similarity [%s,%s]'%(one,two),'=',round(sim(one,two,H),4))
        #if user chose BST
        if input==1:
             print('similarity [%s,%s]'%(one,two),'=',round(simBST(one,two,H),4))
#gets dot product by multiplying the embeddings in position i of both words
def dotProduct(e0,e1):
    tot = 0
    for i in range(len(e0)):
        tot = tot +( e0[i]*e1[i])
    return tot
#gets magnitude of the given embedding(ex)
def Magnitude(ex):
    return math.sqrt(dotProduct(ex,ex))


#comoputes similairty of the given two words
def sim(w0,w1,H):
    e0 = FindC(H,w0)
    e1 = FindC(H,w1)
    dProduct = dotProduct(e0,e1)
    Me0 = Magnitude(e0)
    Me1 = Magnitude(e1)

    result = (dProduct)/(Me0*Me1)
    return result


def simBST(w0,w1,T):
    e0 = Find(T,w0)
    e1 = Find(T,w1)
    dProduct = dotProduct(e0,e1)
    Me0 = Magnitude(e0)
    Me1 = Magnitude(e1)

    result = (dProduct)/(Me0*Me1)
    return result

######################################################
```

```
number = int(input('Choose table implementation:\nType 1 for binary search tree or 2 for hash table
with chaining: '))

print('Choice', number)

print('')

if number == 1:

    print('Building binary search tree\nBinary Search Tree stats: ')

    start = time.time()

    BSTBuilder()

    elapsedTime = time.time() - start

    print('Running time for BST query processing',elapsedTime)


if number ==2:

    print('Building hash table with chaining\nHash table stats: ')

    start = time.time()

    #comparingTextFile(H,2)

    hashTableBuilder()

    elapsedTime = time.time() - start

    print('Running time for hash table query processing',elapsedTime)

if number != 1 and number != 2:

    print('invalid input')
```

Contract:

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 inappropiate assistance to any student in the class.

X [illegible]