Armando Ortiz

April 8, 2019

CS 2302 Data Structures

Lab Report 5

UTEP

Instructor: Olac Fuentes

**Introduction**

The purpose of this lab was to take a text file of words and embeddings created by the glove project and place them in either a binary search tree or hash table with chaining, and then use this data structure to compare the values of words from a separate text list. Only the binary search tree half of this lab was completed.

**Proposed Solution Design and Implementation**

The first problem was figuring out how to read the text from the glove file and place it in a BST. The file was read and then sent to a separate function to create the BST. The function reads through the lines of a text file, and then does a check if each line begins with a letter in the alphabet. The line is then split up, with the embeddings being placed in an array. The array was then converted from string to float. Then, the word and float array are inserted into the tree as a list. This is done with each line in the file. The number of nodes are then calculated by using recursion and going through the tree, adding one as the tree is being moved through.

The other big problem was comparing two words in another text list using the tree. The list was read and split using a loop the same way as the first document, but the strip Python command was used to remove the \n embedding from the second word. These two words were then sent to another function which uses the find function to find each word and their embeddings in the tree. Then the dot product and magnitudes of the their embedding numbers are found and used to find the similarities.

The main possible source of errors in this program is in reading the document. There are several try-except arguments used mainly to ensure that the user is notified if there was trouble reading the documents. A try-except is also used in case the list of words is not formatted correctly or if any of the words are not found in the tree. Also, the outputs for the comparisons and runtimes were rounded to the fifth decimal were rounded to keep the terminal output simple.

**Experimental Results**

These are only the results for a Binary Search Tree. The Hash Table function was not completed. The outputs were ran using a 2017 Macbook Pro, and Spyder.

**Output**

**A close up of text on a white background

Description automatically generated**

**Runtimes**

|  |  |  |
| --- | --- | --- |
| **Comparison List Size** | **Building List** | **Comparisons** |
| **20** | 34.86048 | 0.01394 |
| **30** | 33.28502 | 0.01379 |
| **40** | 38.10394 | 0.01013 |
| **50** | 34.06257 | 0.01622 |

**Conclusion**

One of the most important aspects of using a binary search tree to read files is making sure it runs efficiently. The glove document was very long, and small mistakes could add up, making the runtime very slow. Once the information was inserted into the tree, moving through it was much faster. It shows just how much a structure like a binary search tree or hash table(when completed) can assist with moving through huge collections of information.

**Appendix**

"""

Author: Armando Ortiz

Course: CS 2302 Data Structures

Assignment: Lab 5

Instructor: Olac Fuentes

T.A.: Anindita Nath, Maliheh Zargaran

Last Modification: April 8, 2019

Purpose: Takes glove text file of words and embeddings and places them in a Binary Search Tree

- Prints the number of nodes in the tree

- Prints height of tree

- Takes another text file with list of words and returns similarities of the word's embeddings

- Provides running times of BST construction and word comparisons

"""

import numpy as np

import time

# Class Function

class BST(object):

# Constructor

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

self.item = item

self.left = left

self.right = right

# BST Functions

# Inserts item into BST

def Insert(T,newItem):

if T is None:

T = BST(newItem)

elif T.item[0] > newItem[0]:#Compares words

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

def Find(T,k):

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

if T is None:

return None

if T.item[0] == k: #Finds word

return T.item

if T.item[0]<k:

return Find(T.right,k)

return Find(T.left,k)

#Creates BST out of list

def createBST(T,f):

for line in f:#While reading file

if line[0].isalpha():#If word starts with alphabet letter

spl = line.split(" ")#Splits line into list

a = np.array(([spl[1:]]))#Creates array for word embeddings

arr = a.astype(np.float)#Parses embeddings to float numbers

T = Insert(T,[str(spl[0]),arr])#Places word in BST

return T

# Compares similarities of embeddings of words

def sim(e0,e1):

dot = np.sum(e0\*e1) #Calculates dot matrix

mag = np.sqrt(np.sum(np.power(e0,2))) \* np.sqrt(np.sum(np.power(e1,2)))#Calculates magnitude

return round(dot/mag,4)#Returns rounded number to 4th decimal.

# Finds words and sends them to sim method

def compareBST(T,j,k):

w0 = Find(T,j)#Find word j

w1 = Find(T,k)#Find word k

if w0 is None or w1 is None:#If word not found

return None

else:

return sim(w0[1],w1[1])#send to similarities function

# Counts number of nodes in the BST

def countNodes(T):

if T is None:

return 0

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

# Finds the hight of the BST

def Height(T):

if T is None:

return 0

else:

left = Height(T.left)

right = Height(T.right)

if left > right:

return left + 1

else:

return right+1

T = None

print("Reading word file to build tree")

try:

f = open('glove.6B.50d.txt',encoding='utf-8')#Open Text File

except:#If file is not found

print("File not found")

f.close()

else:

print("Building Binary Search Tree")

print("")

start = time.time()

T = createBST(T,f)

end = time.time()

f.close()

BSTruntime = end - start

nodes = countNodes(T)

print("Number of nodes:",nodes)

height = Height(T)

print("Height:",height)

print("Running time for binary search tree construction:", BSTruntime)

print()

print("Reading word file to find similarities")

try:

fl = open('listwords.txt',encoding='utf-8')#List of words for comparison

except:#If file not found

print("File not Found")

fl.close()

else:

print("Word similarities found")

start = time.time()

for line in fl:

spl = line.split(" ")

try:

spl[1] = spl[1].strip('\n')#Removes new line embedding

c = compareBST(T,spl[0],spl[1])

end = time.time()

except:

print("Comparison not possible")

else:

print("Similarity",spl, "=",c)

BSTCompTime = end - start

print("Running time for binary search tree query processing:",BSTCompTime)

fl.close()

A screenshot of a cell phone

Description automatically generated