Yahir Rivas

Lab 4 Report

Instructor: Dr. Olac Fuentes

Teaching Assistant: Nath, Anindita

CS 2302

MW 1:30 - 2:50

**Lab 5 Report**

Lab 5 consists of implementing two different data types: Binary Search Tree and Hash Table to compare similarities between two words selected from a list of words. Then compare the running times of both implementations and see the differences between both algorithms.

First, I built the binary search tree, I used the methods used for lab 3 and modified it to be able to receive a string instead of an integer by taking newItem as a parameter. However, I was unable to store the file in the binary search tree since my program had issues making the word comparisons.

The way I tried to implement the binary search tree was by reading every line in the file and then storing every word as a string and insert it to the tree. Then I tried comparing the similarity by reading every word from the second text file and comparing it to the next one by finding the embedding from those words. I used the formula provided in the lab description which was the multiplication of the two embeddings divided by distance of the two embeddings. However, when trying to make these comparisons I got the following error: **TypeError: unsupported operand type(s) for \*: 'float' and 'NoneType'**

Index

# -\*- coding: utf-8 -\*-

"""

Created on Thu Mar 28 10:34:37 2019

@author: Yahir F. Rivas

"""

import numpy as np

import time

import math

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([])

class BST(object):

# Constructor

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

self.item = item

self.left = left

self.right = right

self.isLeaf = isLeaf

#insert word into Tree

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

#find the height of the tree

def findheight(T):

if T is not None: #base case

return (1+max([(findheight(T.left)), findheight(T.right)]))

else:

return -1

#return the num of nodes in the tree

def countnodes(T):

if T is not None:

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

return 0

#search for a string in the tree

def search(T, k):

temp = T #temporary variable for T

while temp is not None: #iterate through necessary nodes

if temp.item[0] == k: #found

temp.item[1]

return temp.item[1]

elif temp.item[0] > k: #smaller

temp = temp.left

else: #larger

temp = temp.right

return None #not found

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\*n + ord(c))% n

return r

H = HashTableC(11)

A = ['data','structures','computer','science','university','of','texas','at','el','paso']

for a in A:

InsertC(H,a,len(a))

print(H.item)

for a in A: # Prints bucket, position in bucket, and word length

print(a,FindC(H,a))

def totalkeys(H):

counter = 0

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

counter += 1

return counter

def readTree(): #reads the file with the binary tree

T=None

file=open('glove.6B.50d.txt',encoding='utf-8')

for i in file:

s=i.split(' ') #splits each line

Insert(T, [s[0],np.array(s[1:],dtype=float)]) #inserts

def bst(f, f2):

start = time.time()

T = None

print("Building binary search tree.")

for line in f: #for every line

word = line.split(' ') #separate by ' '

T = Insert(T, [word[0], np.array(word[1:],dtype=float)])

end = time.time()

print("Binary Search Tree stats:")

print("Number of nodes: ")

print(countnodes)

print("Height: ")

print(findheight(T))

print("Running time for binary search tree: ")

print(end - start)

start1 = time.time()

for line2 in f2:

word2 = line2.split(',') #separate every word after a comma

st = search(T, word2[0])

str1 = search(T, word2[1])

print("Similarity", word2[0:2], " = ", round(np.sum(st\*str1)/(math.sqrt(np.sum(st\*st))\*math.sqrt(np.sum(str1\*str1))),4)) #compute the similarity

end1 = time.time()

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

print (end1 - start1)

answer = input(" Choose '1' for Binary Search Tree or '2' for Hash Table: ")

file = open('glove.6B.50d.txt', encoding='utf-8')

words = open('words.txt', encoding='utf-8')

if answer == '1':

print("Binary Search Tree")

bst(file, words)

file.close()

words.close()