Samuel Chong

CS 2302

Olac Fuentes

Lab 5

Introduction:

The purpose for this lab was to compare two words and determine how similar the words are using binary search tree or hash table, depending on the user’s choice.

Unfortunately, I was unable to solve this lab because the program was not reading the files correctly, but this lab report is to explain my thinking and the methods I use, some methods were provided by the professor and some methods I was able to reuse from previous labs (lab from binary search tree).

Binary Search Tree:

In order to read the file, we must use a loop that reads line by line from the file. Since the program is reading everything as a string, then we must split the words with a space. Once the word has been separated, we use the Insert method provided by the professor to insert to the tree. Since we are scanning 2 files, then we must repeat the same process but now with the other file, since my file was separated with commas, then we split the words by a ‘,’. Although I was unable to succeed on reading the files, that was my logic on reading the file since on cs2 we did that using a loop and the condition was “while hasNextLine” and also separate words by a space, so that was my logic on using the loop and separating the words.

numNodes:

If the tree is empty then return 0 because it does not have any nodes. As long as the tree is not empty, then traverse the tree, if there is a node then add 1 and go recursively left and right until reaching the end.

getHeight:

if the tree is empty then return 0. Then create a count to keep track of the levels of the tree. As long as the tree is not empty then we traverse the tree, if it is not empty, then add 1 to the counter. If the left side is not empty then traverse through the left side and if the right side is not empty then traverse through the right side. After reaching the end, we return the counter.

Hash Table:

In order to read the file, I used the same mentality as the binary search tree, go line through line and split the string into words by a space and a comma. Since I used the Insert method provided by the professor in the binary search tree, then I tried using the InsertC method that was provided for us, but I was unable to figure it out how to use it due to the third parameter. For the total elements I used the num\_items in the constructor and the table size is the length of the hash table. Then compute the similarity of the words with the formula.

In the “main method” it is supposed to read and open the files, and the user chooses whether they want to use binary search tree or hash table for similarity of words method.

Code:

﻿#!/usr/bin/env python3

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

"""

Created on Wed Apr 10 20:32:32 2019

@author: sachong

"""

import numpy as np

import time

import math

# Implementation of hash tables with chaining using strings

####################### 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 = []

self.num\_items = 0

for i in range(size):

self.item.append([])

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 mainHash(F, F2):

print("Building hash table.")

H = HashTableC(15) #create size 15

print("Initial table size: ")

print(len(H.item))

for line in F: # read line by line, glove

word = line.split(' ')

#H = InsertC(H, word) # insert data using the InsertC method

print("Total elements: ")

print(H.num\_items)

print("Final table size: ")

print(len(H.item))

print("Load factor: ")

print( H.num\_items/len(H.item))

start = time.time()

for line2 in F2:

word2 = line2.split(',')

st = FindC(H, word2[0])

str1 = FindC(H, 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

end = time.time()

print("Running time for hash table query processing: ")

print(end-start)

####################### BINARY SEARCH TREE ###############################

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 Find(T,k):

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

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

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

def wordSearch(T,k):

temp = T

while temp is not None: #go through the tree

if temp.item[0] == k: #if the items are the same

temp.item[1]

return temp.item[1]

elif temp.item[0] < k: #if the item is larger then go rigtht

temp = temp.right

else: #else go left if smaller

temp = temp.left

return None

def getHeight(T):

if T is None: #if it is empty return 0

return 0

count = 0

current = T

while current is not None: #while the tree is not empty

count = count + 1 #add 1 everytime it iterates

if T.left is not None: # if the left side is not None then go left

current = current.left

elif T.right is not None:#if the right side is not None then go right

current = current.right

return count #return the counter

def numNodes(T):

if T is None:

return 0

if T is not None:

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

return 0

def bst(F, F2):

start = time.time()

T = None

print("Building binary search tree.")

for line in F: #go line by line

word = line.split(' ')

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

end = time.time()

print("Binary Search Tree stats:")

print("Number of nodes: ")

print(numNodes)

print("Height: ")

print(getHeight(T))

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

print(end - start)

start1 = time.time()

for line2 in F2:

word2 = line2.split(',') #since the words are separated by a ,

st = wordSearch(T, word2[0])

str1 = wordSearch(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')

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

if answer == '1':

print("You have chosen the Binary Search Tree")

bst(file, myfile)

elif answer == '2':

print("You have chosen Hash Table")

mainHash(file, myfile)

file.close()

myfile.close()

I, Samuel Chong, sign the academic honesty certification. This is my work and only my work. No external help was used for this lab. Also, this report was made by me and no collaboration was made.