

Sergio Gramer – 88521512
Lab #4: B-Trees

In lab 5 we are tasked with Natural Language Processing. We are given a list of words to compare to each other and with a file provided by Stanford University, we compare them. By opening the file in Spyder using Python 3, we save the file as either A) a hash table or B) a Binary Search tree. From there we open a file that we created of words that we would like to compare. We begin saving line by line of the second file and sending them to either the Hash Table or the binary search tree to look for the words and get the numbers associated with that word to use the formula provided in the Lab 5 pdf to calculate the similarities.

I began the lab by learning how to open files in python. At first I was opening the file as with `open(filename, "r")` but for some reason I kept getting errors. I then changed the opening procedure encoding to UTF-8 and got no errors. After this I had to gather each item in the list and save the first part of it (the word itself) into an array along with the numbers associated with it. I did this by splitting the line after each space and saving the first part as a string and then using the numpy to save as an array of floating point numbers afterwards. The biggest challenge I faced was getting the words that I had chosen from the file and only retrieving the numbers that it was associated with. To do this, I modified the original and given FindC function to return the item at the calculated height or index of the hash at whatever value the current word equaled with the second part of it (not to include the word) saved as the second index of the array. I then just used numpy to calculate the dot products and magnitudes.

Using the same words as the professor to compare results with a binary search tree

```
Type "hash" for Hash Table or "bst" for Binary Search Tree

bst
Chose Binary Search Tree
Similarity of bear and bear is :
0.9999999999999999
Similarity of barley and shrimp is :
0.5352692835187685
Similarity of barley and oat is :
0.6695943873520417
Similarity of feather and baseball is :
0.2091094209768219
Similarity of federer and tennis is :
0.7167607558215532
Similarity of harvard and stanford is :
0.8466463005377867
Similarity of harvard and utep is :
0.06842559497545814
Similarity of harvard and ant is :
-0.026703801487502007
Similarity of raven and crow is :
0.615012230558943
Similarity of raven and whale is :
0.3290891478461917
Similarity of spain and france is :
0.7909148774160825
Similarity of spain and mexico is :
0.7513764510070793
Similarity of mexico and france is :
0.5477963355783697
```

Using my own set of words to find the results of the words with a BST :

```
Similarity of utep and stanford is :
0.29289160141577797
Similarity of phone and cell is :
0.6038961152058904
Similarity of screen and monitor is :
0.5289487814564959
Similarity of coffee and energy is :
0.47255925226280737
Similarity of woman and baby is :
0.7111113810903866
Similarity of baby and milk is :
0.5843765593196951
Similarity of paper and pencil is :
0.6001403107253652
Similarity of house and car is :
0.4729318862998191
Time it took to build BST: 21 seconds.
Height of the BST: 52
Number of nodes in BST: 400000
```

Using the professors words to compare results with a hash table :

```
Type "hash" for Hash Table or "bst" for Binary
Search Tree

hash
Chose Hash Table
Similarity of bear and bear is :
0.9999999999999998
Similarity of barley and shrimp is :
0.5352692835187685
Similarity of barley and oat is :
0.6695943873520417
Similarity of feather and baseball is :
0.2091094209768219
Similarity of federer and tennis is :
0.7167607558215532
Similarity of harvard and stanford is :
0.8466463005377867
Similarity of harvard and utep is :
0.06842559497545814
Similarity of harvard and ant is :
-0.026703801487502007
Similarity of raven and crow is :
0.615012230558943
Similarity of raven and whale is :
0.3290891478461917
Similarity of spain and france is :
0.7909148774160825
Similarity of spain and mexico is :
0.7513764510070793
Similarity of mexico and france is :
0.5477963355783697
Similarity of mexico and guatemala is :
0.8113810916121009
```

Using a set of words that I chose to find their similarities with hash table:

```
Similarity of utep and stanford is :
0.29289160141577797
Similarity of phone and cell is :
0.6038961152058904
Similarity of screen and monitor is :
0.5289487814564959
Similarity of coffee and energy is :
0.47255925226280737
Similarity of woman and baby is :
0.7111113810903866
Similarity of baby and milk is :
0.5843765593196951
Similarity of paper and pencil is :
0.6001403107253652
Similarity of house and car is :
0.4729318862998191
Time it took to build tables: 18 seconds.
Initial Table Size: 69
Final Table Size: 573439
Percentage of empty lists: 99.93198927872015
Standard deviation of the lengths of the lists:
1.8444410350723872e-12
```

In conclusion I learned that it takes longer to build a binary search tree than a hash table for this project. It is better to use a hash table because you can traverse it easily. Also, there are great uses for this. I learned that using different data structures for different things some are better and some are worse at doing different types of projects. I saw that using a hash table might be trickier to write in code but ends up being better for these types of projects.

Appendix: Source Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 7 13:41:32 2019

@author: SergioGramer
Class: CS2302
Instructor: Dr. Olac Fuentes
TA: Anindita Nath
Lab : 5
"""
import numpy as np
import math
import time
# Implementation of hash tables with chaining using strings

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):
    if H.num_items // len(H.item) == 1:
        H = DoubleLoadFactor(H)
    b = h(k[0], len(H.item))
    H.item[b].append([k[0], np.array(k[1:]).astype(np.float)])
    H.num_items += 1
    return H

def DoubleLoadFactor(H) :
    H2 = HashTableC((len(H.item) * 2) + 1)
    for i in H.item :
        if i is not [] :
            for j in i :
                H2.item[h(j[0], len(H2.item))].append([j[0], j[1]])
            H2.num_items += 1
    return H2

def FindC(H,k):
    b = h(k,len(H.item))
    for a in range(len(H.item[b])):
        if H.item[b][a][0] == k:
            return H.item[b][a][1]
    return -1

def h(s,n):
    r = 0
    for c in s:
        r = (r * n + ord(c)) % n
    return r

#####
#BST#
#Copy/Pasted from lab 3
#####

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[0] > newItem[0]:
        T.left = Insert(T.left, newItem)
    else:
        T.right = Insert(T.right, newItem)
    return T

def height(T) : #finds the height of a BST
    if T is None :
        return -1
    else :
        l = 1 + height(T.left) #calculates height of left side + 1 because we already passed root
        r = 1 + height(T.right) #calculates height of right side
    if l >= r : #if left side is greater or equal send left, no need to go further if equal
        return l
    else : #else right is greater send right
        return r

def Search(T, k) : #Searches a BST 'T' for item "k"
    while T is not None : #Ensures we don't traverse and empty BST
        if T.item[0] == k :
            return T.item[1]
        elif T.item[0] > k : #Compares the number are looking for to the item we are
            T = T.left #currently at to see if it is greater, goes to left
        else : #else goes to the right side
            T = T.right
    return None

def numberOfNodes(T) : #Will count number of number if BST (T) is not none
    if T is not None :
        return 1 + numberOfNodes(T.left) + numberOfNodes(T.right) #Adds 1 because of root.
    return 0

#####
#RUN#
#####

print('Type "hash" for Hash Table or "bst" for Binary Search Tree')
ans = input() #will save the input as ANS
if ans == "hash" :
    InitialTable = 69
    print("Chose Hash Table")
    TimeStart = int(time.time()) #begins counting the time it takes to build the hash table
    H = HashTableC(InitialTable)
    with open("glove.6B.50d.txt", encoding='utf-8') as file : #opens the glove file as
        for line in file : #goes line by line in the file
            word = line.split(' ') #saves them as word splitting them every space
            H = InsertC(H, word) #inserts the word into the HashTable

    with open("similar.words.txt", encoding='utf-8') as file2 : #opens the second file that i have the words saved in
        for line2 in file2 : #goes line by line
            word2 = line2.split() #saves the lines in word splitting them by space
            e0 = FindC(H, word2[0]) #searches the first word in the hash table and saves the numbers as e0
            e1 = FindC(H, word2[1]) #searches the second work in the hash table and saves the numbers as e1
            TimeEnd = int(time.time())
            print('Similarity of ', word2[0], ' and ', word2[1], ' is :')
            print((np.sum(e0 * e1) / (math.sqrt(np.sum(e0 * e0)) * math.sqrt(np.sum(e1 * e1))))))
            print('Time it took to build tables: ', TimeEnd - TimeStart, ' seconds.')
    count = 0
    m = H.num_items / len(H.item)
    k = 0
    for a in H.item:
        k += len(a) - m
        if a is [] : #adds one if finds an empty list
            count += 1
    d = (1 / len(H.item)) * k

```

```

print('Initial Table Size: ', InitialTable) #prints initial hash table size
print('Fnal Table Size: ', len(H.item)) #prints ending hash table size
print("Percentage of empty lists:", count / len(H.item) * 100) # divides count by the length *100 to get percentage of empty lists
print("Standard deviation of the lengths of the lists:", d)

elif ans == "bst" :
    print("Chose Binary Search Tree")
    TimeStart = int(time.time())
    T = None
    with open("glove.6B.50d.txt", encoding='utf-8') as file :
        for line in file :
            word = line.split()
            T = Insert(T, [word[0], np.array(word[1:]).astype(np.float)])

    with open("similar.words.txt", encoding='utf-8') as file2 :
        for line2 in file2 :
            word2 = line2.split()
            e0 = Search(T, word2[0])
            e1 = Search(T, word2[1])
            TimeEnd = int(time.time())
            print('Similarity of ', word2[0], ' and ', word2[1], 'is :')
            print((np.sum(e0 * e1) / (math.sqrt(np.sum(e0 * e0)) * math.sqrt(np.sum(e1 * e1)))))
            print("Time it took to build BST: ", TimeEnd - TimeStart, ' seconds.')
            print('Height of the BST: ', height(T))
            print('Number of nodes in BST: ', numberOfNodes(T))

    else :
        print("Try again. \nPlease type \"bst\" or \"hash\".")

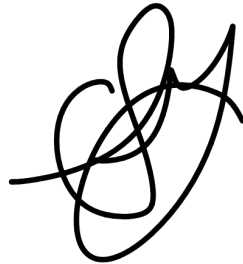
```

ACADEMIC CERTIFICATION

I “Sergio Gramer” 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.

Signed: 04/12/2019

Sergio Gramer

A handwritten signature in black ink, consisting of several loops and a final vertical stroke, positioned above a horizontal line.