Luis Renteria

ID 88740232

CS 2302

Dr. Fuentes

Lab Report 5

**Introduction**

The object of our fifth lab assignment was to utilize a hash table and binary search tree, as well as a text file to create a list of words to compare and determine the similarity of. The tasks that we had to accomplish were reading the file and splitting it accordingly, organizing the data into a hash table or BST, and finally, reading a file and calculating the similarity between the two words.

Note: I was not able to finish this assignment and will therefore explain the tasks as best as I can, in the way that I understood them.

**Reading the Glove File**

Before any calculations would be made, we had to read a file which I will refer to as the “glove file”. The glove file contained over 400,000 words, as well as a list of 50 embeddings for each word. Each word along with the 50 embeddings had to each be put into a single node, which would in turn be inserted into the hash table. To read this file, I used a for loop to iterate through each line of the file, and stored each line in a variable called “line”. In order to split the line from the glove file, I used the regEx function of split, and separated the word from the 50 embeddings. I then inserted every embedding into a size 50 numpy array and returned both the word and filled array. Once the word and accompanying array were returned, I used the provided InsertC method to insert the two variables into a hash table or a binary search tree, depending on which option the user chose.

**Reading Word Pair File**

Before being able to find the similarities between different words, we had to first create different words to compare. Using a method InsertList, I read a word file line by line using re.split and stored it in a variable, line. From line, I inserted the strings into a list containing lists of size two. After creating the list, I return it.

**Calculating Similarity**

In order to calculate the similarity of two words, we were given an equation that multiplies the embeddings of each word together, then divides it by the absolute value of the embeddings. The FindSimilarities method prints out the statistics for the hash table/BST and each pair of words and their similarity on a range of 1 to -1, -1 being the worst and 1 being the best.

**Conclusion**

In this lab I was able to see a real-world application of computer science. Although I was not able to obtain results, the results that were shown to us in class was very interesting. It surprised me how accurate the results were even though the computer had no understanding of the words themselves.

**Academic Statement**

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 inappropriate assistance to any student in the class.

Signed, Luis Renteria.

**Appendix**

This was the code I used from the class website, as provided by Dr. Fuentes.

# 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 = []

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

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))

I also used some of the b-tree code, which I will now insert here.

# Code to implement a binary search tree

# Programmed by Olac Fuentes

# Last modified February 27, 2019

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)