Lab 5 for this semester was by far the hardest lab. The lab is about the Natural Language Processing (NLP) which is a su-field to artificial intelligence that deals with designing algorithms that can understand the human language. In short we where to recreate this in a much smaller scale but that did not decrease the difficulty of it. For this lab we have to design a hash table with chaining and a binary search tree that would store information from ‘glove.6B.50d.txt’. A file that contain a bunch of words and their vectors of numbers.

The first step to this lab was to prompt the user to choose the option of ether use a hash table or a binary search tree. Using a simple input() function with an if else statement to make a decision solves this part and that is exactly what I did. After that is done we create our binary search tree and hash table depending on the option they choose. Up to this point the lab was very simple.

The next part of the lab was also relatively pretty simple. The next part was to read the file ‘glove.6B.50d.txt’ and store it in its correct data format. To read the file I first had to open the file by using the open() function and pass the name of the file and the letter ‘r’ to tell the computer that I want to open the file and read it only. After opening the file in read mode i had to make it so that I read each line of information separately instead of all the information together to make it easier to deal with. For this I use readlines() and store it in a variable called contents. This basically creates a list that stores each line from the file. After that is done using a for loop to traverse each line of information we then proceeded to split each line into sections. Using the split() function to separate the line by black spaces we now have each part of the line in a list. This way we can get the word and the vectores and inserted them into our data structures. This is basically the way I read my file but it only go hard from here because from here i had to inserted them into the hash table or the binary tree.

Inserting the information that I had just read and separated was the hardest part of this lab. The reason been that Im only used to storing integers into ether the hash table or binary search tree and not words. Meaning that I had to rethink in how i would insert the words into by data structures. I first started by creating the function to insert for the binary search tree. I tried to make the easiest possible way to sort and store the info into the binary search tree. Using a function called Insert() I made it take 3 inputs. The tree root, the word , and the list of numbers. Then just like when storing integers I first check if the tree is empty. If the tree is empty we create a new node and set it as our root. Otherwise we check the first character of the word we are trying to input to the first character in the word of the node. Using the greater or lesser symbols to determine which one comes before the other we then traverse the tree until we find an empty spot for it. Next For the hash table was a lot more easier to insert. Apart from inserting we also had to watch for the load factor to make sure there were not that many collisions. So everytime we added something we would also check our load factor and then increase the size of the table if needed. First for the insertion I made the function InsertC() that takes 3 inputs. H the hashtable, W the word, and N the list of numbers. I First past the word and length of table to my function that hashes my word. There we traverse each character of the word and multiply it by 255 add the ord() of the character and get the % by the length of the table. In the end we return the hash number back to the insert function where we Go to the posting of the hash in the table and store the word and the list of numbers at that position as a list and if there is already another word there then we just append the new list to the list. The reason ths was the hardest part to me was because as I had a short time to finish lab 5 I was not able to solve a problem that I had when adding the information to the binary tree as I had an infinite loop somewhere and didn't know how to fix it.

The last part to this lab was to find the similarity between words or in other words the embeddings. For this part I made a function called similarity() that takes 3 inputs. T or H that are the binary tree or the hash table, word1 that is the first word I'm looking for, and word2 thats the second word im looking for. For the hash table and the binary search I need to create a find function that follow the traversal that insertion() uses. This was even harder as i was already having trouble inserting the information because of the traversal I was using. Basically using the insertion method I created a function for the hash and binary tree that follow the exact same steps and compared the words to see if they matched if they did return the number, if they did not find the word then they return not found. After that i got the first part of the function to find the similarity that is to multiply and add the product of the two sets of number list. After I had to find the square root of adding the two sets of list. In the end I return the division of the multiplication part divided by the square root of the addition part.

In conclusion after running my program bouth data structures had problems. The binary search tree was not able to insert the information because of a infinite loop in the function. In the hash table everything went okay till it was finding the similarity of the words where my find function was not working and return word not found for all words. This problem might have arised from my laptop not having enough memory to read the complete file as every time I tried to open the file it would only show a part of it and warned that the finale was to big. I believe given more time I can solve the two problems but for now the program does not do what it was meant to do.

# Lab 5

# Programmed by Anthony Herrera

# Last modified May 8, 2019

import math

class BST(object):

# Constructor

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

self.item = item

self.left = left

self.right = right

def Insert(T,word,numbers,ind = 0):

if T == None:

T = BST([word,numbers])

elif T.item[0][ind] > word[ind]:

T.left = Insert(T.left,word,numbers,ind)

else:

T.right = Insert(T.right,word,numbers,ind)

return T

def Find(T,word):

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

return T.item[1]

if T.item[0][0] > word[0]:

return Find(T.left,word)

else:

return Find(T.right,word)

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.count = 0

for i in range(size):

self.item.append([])

def IncreaseC(H):

if H.count // len(H.item) > 1:

temp = HashTableC(len(H.item)\*2 +1)

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

for y in H.item[x]:

temp.item[x].append(y)

return temp

return H

def InsertC(H,W,N):

b = h(W,len(H.item))

H.item[b].append([W,N])

H.count += 1

def h(W,n):

r = 0

for c in W:

r = (r\*255 + ord(c)) % n

return r

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][1]

return False

def Similarity(T,word1,word2):

e0 = Find(T, word1)

e1 = Find(T, word2)

if e0 == False or None and e1 == False or None:

return 'Word not Found'

mult = 0

for x in e0:

for y in e1:

mult += x \* y

absolutV = 0

for x in e0:

for y in e1:

absolutV += x + y

return mult / math.sqrt(absolutV)

def SimilarityC(H,word1,word2):

e0 = FindC(H,word1)

e1 = FindC(H,word2)

mult = 0

if e0 == False or None and e1 == False or None:

return 'Word not Found'

for x in e0:

for y in e1:

mult += x\*y

absolutV = 0

for x in e0:

for y in e1:

absolutV += x+y

return mult / math.sqrt(absolutV)

option = input('Choose a table implementation binary search tree(1) or hash table chaining(2)\n')

if option == '1':

T = None

counter = 0

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

contents = file.readlines()

for x in contents:

line = x

line\_split = line.split()

T = Insert(T,line\_split[0],line\_split[1:])

counter += 1

file.close()

print('Number of Nodes: ', counter)

print('Similarity [bear,bear]', Similarity(T,'bear', 'bear'))

print('Similarity [barley,shrimp]', Similarity(T,'barley', 'shrimp'))

print('Similarity [barley,oat]', Similarity(T,'barley', 'oat'))

print('Similarity [federer,baseball]', Similarity(T,'federer', 'baseball'))

print('Similarity [federer,tennis]', Similarity(T,'federer', 'tennis'))

print('Similarity [harvard,stanford]', Similarity(T,'harvard', 'stanford'))

print('Similarity [harvard,utep]', Similarity(T,'harvard', 'utep'))

print('Similarity [harvard,ant]', Similarity(T,'harvard', 'ant'))

print('Similarity [raven,crow]', Similarity(T,'raven', 'crow'))

print('Similarity [raven,whale]', Similarity(T,'raven', 'whale'))

print('Similarity [spain,france]', Similarity(T,'spain', 'france'))

print('Similarity [spain,mexico]', Similarity(T,'spain', 'mexico'))

print('Similarity [mexico,france]', Similarity(T,'mexico', 'france'))

print('Similarity [mexico,guatemala]', Similarity(T,'mexico', 'guatemala'))

print('Similarity [computer,platypus]', Similarity(T,'computer', 'platypus'))

else:

H = HashTableC(11)

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

contents = file.readlines()

for x in contents:

line = x

line\_split = line.split()

H = IncreaseC(H)

InsertC(H,line\_split[0],line\_split[1:])

file.close()

print('Initial table size:', 11)

print('Final table size:',len(H.item))

print('Similarity [bear,bear]', SimilarityC(H,'bear', 'bear'))

print('Similarity [barley,shrimp]', SimilarityC(H,'barley', 'shrimp'))

print('Similarity [barley,oat]', SimilarityC(H,'barley', 'oat'))

print('Similarity [federer,baseball]', SimilarityC(H,'federer', 'baseball'))

print('Similarity [federer,tennis]', SimilarityC(H,'federer', 'tennis'))

print('Similarity [harvard,stanford]', SimilarityC(H,'harvard', 'stanford'))

print('Similarity [harvard,utep]', SimilarityC(H,'harvard', 'utep'))

print('Similarity [harvard,ant]', SimilarityC(H,'harvard', 'ant'))

print('Similarity [raven,crow]', SimilarityC(H,'raven', 'crow'))

print('Similarity [raven,whale]', SimilarityC(H,'raven', 'whale'))

print('Similarity [spain,france]', SimilarityC(H,'spain', 'france'))

print('Similarity [spain,mexico]', SimilarityC(H,'spain', 'mexico'))

print('Similarity [mexico,france]', SimilarityC(H,'mexico', 'france'))

print('Similarity [mexico,guatemala]', SimilarityC(H,'mexico', 'guatemala'))

print('Similarity [computer,platypus]', SimilarityC(H,'computer', 'platypus'))

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion. Cheating may involve copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data (for example program outputs) in laboratory reports. Plagiarism occurs when someone represents the work or ideas of another person as his/her own. Collusion involves collaborating with another person to commit an academically dishonest act. Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

* Anthony Herrera