Lab 5: Natural Language Processing Lab

Problem:

We were given a file with words and vectors. We need to build a binary search tree and a hash table with chaining, find the similarities, compare the both structures.

Proposed Solution:

First ask the user if he wants to use a binary search tree or a hash table. Depending on what the user inputs. You would create a binary search tree or a hash table. It is more efficient to compare the words in a hash table than in a binary search tree because you can find words in constant time O(1). In a binary search tree you at least need O(logn) to traverse. The similarity you need to use the dot product and the msgnitude divide them and then that is your answer.

Conclusion:

I learned that the hash table is more efficient in accessing data because it can do it in constant time. But if you are inserting it is better to use a binary search tree because you traverse only half the tree.

Appendix:

import numpy as np

class BST(object):

# Constructor

def \_\_init\_\_(self, item, vector,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

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\*n + ord(c))% n

return r

def build\_binary(L):

T = None

for line in word:

w = line.split(" ")

if w.isalpha():

vector = np.array(50)

for i in range(1,len(w)):

vector[i-1] = float(w[i])

T.Insert(T,[[w[0]][vector]])

return T

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))

table = input("Do you want to use a hashtable or a binary-search-tree?")

if table == "hashtable":

ht = open("glove.6B.50d.txt",'r')

print("Hello World!")

elif table == "binary-search-tree":

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

word = bst.readlines()

build\_binary(word)

else:

print("not a valid table")

Academic Agreement:

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.

Andres Arellanes