Lab 5

Fernando De Santiago

University of Texas at El Paso

Introduction

The purpose of this lab is to input a file and create a hash table and BST. The lab is broken into 5 parts but not all of them were completed due to knowledge constraint. The first requirement wasn’t really a method but instead just creating a choice statement. The second requirement is to read a txt file and insert it into a hash and BST. The hash requires it to be with chaining and string manipulation the BST was required to have the words and embeddings put together. The third requirement consisted of displaying statistics describing the Hash Table and BST Implementation. The fourth requirement which I was not able to meet was to create a method that would get the similarities between words. This would have been done using ascii values. The 5rh was to just measure run times which I was unable to do with the similarities since I wasn’t able to run it

# Requirement 1

The first requirement was to create a choice for the user to make. After this I began to important the methods needed that were used in previous methods. After I started two method that would build the BST and the Hash.

## Requirement 2

The requirement had me start and construct 2 different building methods. I had some problems with reading the file sometimes that would cause the file to crash my computer as well as have the embeddings in scientific notation and word would display \n but cause problems. I also created the load factor method needed in order to have it checked till I reached a load factor less than 1.

### Requirement 3

This was the easiest to make since it just required to print stuff out and display the statistics. While doing this I found which methods were and weren’t working by commenting stuff out. This was the go to in order to test my methods one at a time. I ran into a problem with percent empty at first since it was doubling the size of the empty array’s so I made it divide by 2 before returning it. I also had a big mistake using the standard deviation that I wasn’t able to fix before the time I needed to turn this in. I tried using np.std but nothing so I eventually gave up.

#### Requirement 4

For this requirement I didn’t wasn’t able to finish it but I will talk about how I was planning to implement it. I planned for the hash table to create a method that would take the embeddings and make them float values while having them multiply it by the embedding for the other words. The other way I wanted to do it but didn’t have the time was to us the np.dot. also wanted to use the other numpy tool which would do the magnitude then I would get the two embeddings multiplied and then divided it by the magnitude. For the BST I would of tried to convert the letters into ascii values and then put them into the BST. I tried to code it but just couldn’t figure out how to make it work.

##### **Requirement 5**

For this requirement I just ran the time function needed but was unable to have them for the similarities.

**Conclusion**

This lab was the hardest I have had to do and even with proper time. I tried to meet as many requirements but I couldn’t and ran into problems by not understanding how to convert the strings in the file into float values. Thus I was unable to finish the last 2 requirements.

**Appendix**

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

**"""**

**Created on Sun Mar 31 20:47:34 2019**

**@author: Fernando**

**"""**

**# Code to implement a HashTable and BST**

**# Programmed by Olac Fuentes**

**# Last modified April 4th, 2019**

**#CS2302**

**#Fernando De Santiago**

**#LAB5**

**#Olac Fuentes, Anindita Nath and Maliheh Zargaran**

**#last edited 3/15/19 21:48:00 PM**

**#Section M/W 10:30-11:50**

**#purpose: to read files with python and import and test values with hash and BST**

**import time**

**import numpy as np**

**from numpy import array**

**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 Find(T,k):**

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

**if T is None or str(T.item[0]) == str(k):#comparing strings**

**return T**

**if str(T.item[0])<str(k):**

**return Find(T.right,k)**

**return Find(T.left,k)**

**def Numbers(T):#counting nodes**

**if T is None:**

**return 0**

**if T is not None:**

**Counter=1+Numbers(T.left)+Numbers(T.right)**

**return Counter**

**def height(T):#gets height**

**if T is None:**

**return 0**

**left=1**

**right=1**

**left+=height(T.left)**

**right+=height(T.right)**

**if left>right:**

**return 1+left**

**return 1+right**

**def BSTTree():#creates BST**

**try:**

**with open("glove.6B.50d.txt",encoding='utf-8') as f:**

**liners=f.readline()**

**T=None**

**while liners:**

**EA=[]#empty array**

**EA=liners.split(" ")**

**if EA[0].isalpha():**

**N=((EA[1:]))**

**T=Insert(T,(EA[0],N))**

**liners=f.readline()**

**return T**

**except:**

**print("File not found")**

**def contrast(T):#used to find similiarties**

**try:**

**with open("pairs.txt",encoding='utf-8') as f:**

**liners=f.readline()**

**counter=0**

**while liners:**

**EA=[]**

**EA=liners.split(" ")**

**Word0= Find(T, EA[0])**

**print(EA[0],EA[1])**

**Word1= Find(T,EA[1])**

**print(Word0)**

**print(Word1)**

**counter+=1**

**dot = np.sum(Word0\*Word1,dtype=float)#tried using the num.sum method**

**print(dot)#tried getting the dot product**

**liners=f.readline()**

**except:**

**print("File not found")**

**#---------------------------------------------------------------------------------**

**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 str(s):#changed to string**

**r = (r\*len(str(s)) + ord(c))%n#getting string and get the ord**

**return r**

**def BuildH():#Creating the Hash Table**

**try:**

**with open("glove.6B.50d.txt",encoding='utf-8') as f:**

**lines=f.readline()**

**words=[]**

**E=[]**

**while lines:**

**A=[]**

**A=lines.split(" ")**

**if A[0].isalpha():**

**words.append(A[0])**

**E.append((A[1:]))**

**InsertC(H,words[0],E)**

**lines=f.readline()**

**return H**

**except:**

**print("File not found")**

**def STD(H):#trying to get the Standard Deviation**

**try:**

**with open("Pairs.txt",encoding='utf-8') as f:**

**lines=f.readline()**

**E=[]**

**while lines:**

**lists=lines.split(" ")**

**if lists[0].isalpha():**

**E.append((lists[1:]))**

**lines=f.readline()**

**a=array(E)**

**return a**

**except:**

**print("File not found")**

**def LoadFactor(H):#creating load factor**

**if H is None:**

**return -1**

**counter=0**

**for i in range(len(H.item)):**

**counter+=len(H.item[i])**

**sums= counter/len(H.item)**

**return sums**

**def Double(H):#doubles size**

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

**for i in range(len(H.item)):**

**for j in range(len(H.item[i])):**

**InsertC(HT,((H.item[i])[j])[0], ((H.item[i])[j])[1])**

**return HT**

**def Percent(H):**

**empty=0**

**for i in H.item:**

**if len(i)==0:**

**empty+=1**

**empty=empty/2**

**return (empty/len(H.item))\*100**

**#------------------------------------------------------------------------------**

**print("please choose a table implementation")**

**print("1. Binary Search Tree")**

**print("2. Hash Table with Chaining")**

**choice=int(input('decision '))**

**if choice==1:**

**print()**

**print("Building Binary Search Tree")**

**T=BSTTree()**

**print()**

**print("Binary Search Tree stats: ")**

**print("number of nodes: ")**

**print(Numbers(T)+1)**

**print("Height: ")**

**print(height(T))**

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

**start=time.time()**

**BSTTree()**

**end=time.time()**

**print(end-start, ' seconds')**

**print("Reading word file to determine similarities")**

**contrast(T)**

**print()**

**print("running time for binary search tree query processing: ")**

**elif choice==2:**

**H=HashTableC(97)**

**H=BuildH()**

**print("Building hash table with chaining")**

**print()**

**print("Hash table stats")**

**print()**

**print("Initial table size: ",len(H.item))**

**while LoadFactor(H)>=1:**

**H=Double(H)**

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

**print("Load factor: ",LoadFactor(H))**

**print("Percentage of empty lists: ",Percent(H))**

**print("Standard deviation of the lengths of the lists: ",STD(H))**

**print()**

**print("Reading word file to determine similarities")**

**print()**

**print("Word similarities found: ")**

**print()**

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

**else:**

**print("invalid")**

**“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.”**

**Fernando De Santiago**