CS 2302

Lab 5 Report

Author: David Ayala

ID: 80617838

Instructor: Olac Fuentes

TA: Anindita Nath

**Introduction**

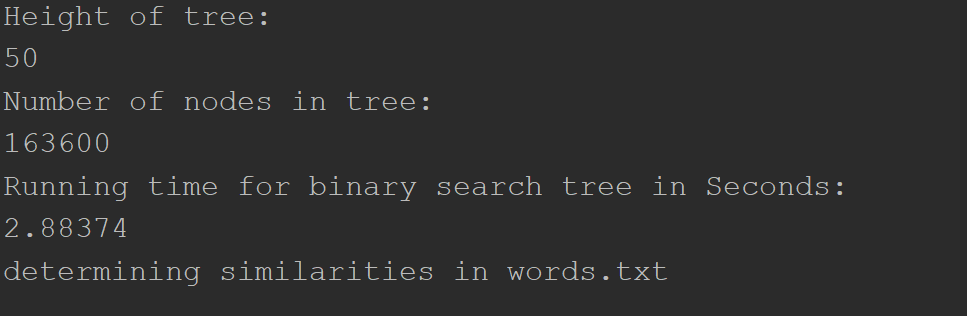
The purpose of this lab was to Prompt the user to choose a table implementation (binary search tree or hash table with chaining). Read the file ”glove.6B.50d.txt” and store each word and its embedding in a table with the chosen implementation. Compute and display statistics describing your hash table. See the appendix for examples for both implementations. Read another file containing pairs of words (two words per line) and for every pair of words find and display the ”similarity” of the words. Display the running times required to build the table (item 2) and to compute the similarities (item 4).

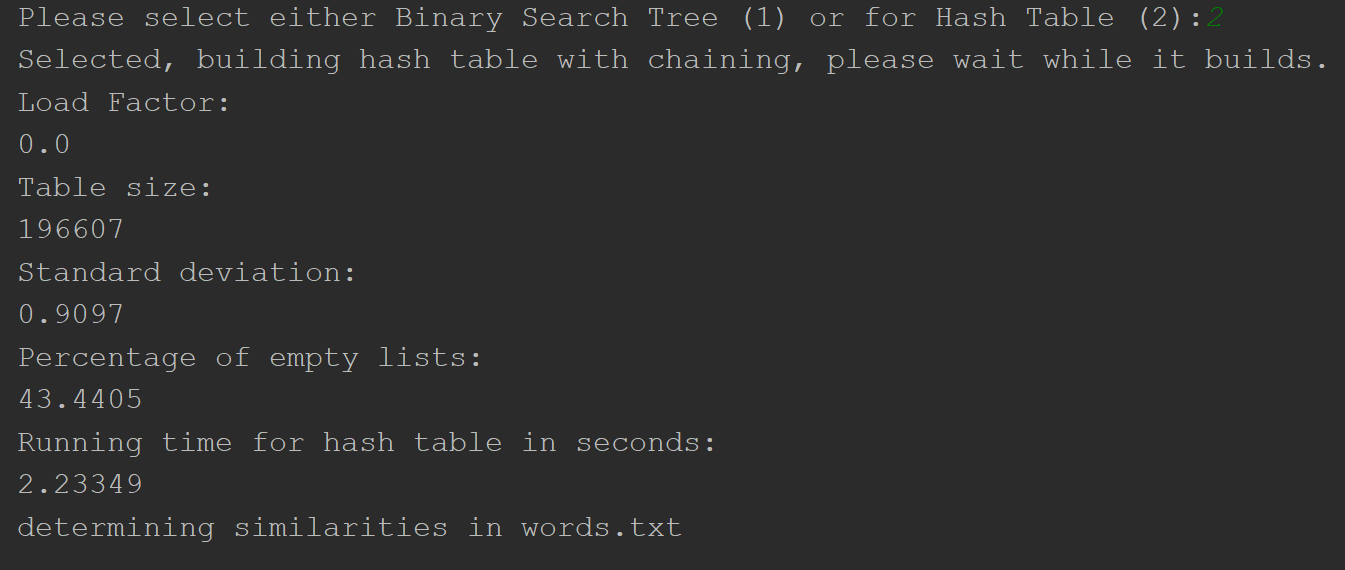
**Proposed Solution & Design Implementation**

so I knew that I would need to find have a BST and a Hash Table so I went used the code that Fuentes put on the website. From there I had had to download the zip file from the website that Fuentes gave us, I then put the Glove file in the same location as the Lab file. One I did I had use think about how to go about programing it, so I knew I would have to read a file so I had to figure out how to do that. I also had to ask the user what they wanted to use (either BST or Hash Table) once that was done I then moved on to figuring what I had to find out what I needed for either choice. I then made a file for all the words I picked.

**Experimental Results**

|  |  |  |
| --- | --- | --- |
| **Size of list** | **BST outputs** | **Hash Table Output** |
| 50 | Height:50  #Nodes:163600  Tree running time:2.88374  Comparison running time:0.0099 | Size: 196607  Empty percent: 43.4405  Standard Deviation: .9097  Load Factor: 1.6834  Hash Construction: 2.23349  Query processing:0.00695 |
| 100 | Height:100  #Nodes: 327200  Tree running time: 5.76748  Comparison running time: 0.0198 | Size: 393214  Empty percent:44.5624  Standard Deviation: .9365  Load Factor: 1.7296  Hash Construction: 4.46698  Query processing: 0.0139 |
| 150 | Height:150  #Nodes: 409000  Tree running time: 8.65122  Comparison running time: 0.0297 | Size: 589821  Empty percent: 45.6849  Standard Deviation: .9492  Load Factor: 1.7245  Hash Construction: 6.70047  Query processing: 0.02085 |
| 200 | Height:200  #Nodes: 654400  Tree running time: 11.53496  Comparison running time: 0.0396 | Size: 786428  Empty percent: 46.5214  Standard Deviation: .9681  Load Factor: 1.8624  Hash Construction: 8.93396  Query processing: 0.0278 |





**Conclusion**

This lab allowed us to learn how to compare different words and to see if they are similar. We did this by Prompt the user to choose a table implementation (binary search tree or hash table with chaining). Read the file ”glove.6B.50d.txt” and store each word and its embedding in a table with the chosen implementation. Compute and display statistics describing your hash table. See the appendix for examples for both implementations. Read another file containing pairs of words (two words per line) and for every pair of words find and display the ”similarity” of the words. Display the running times required to build the table (item 2) and to compute the similarities (item 4).

**Appendix**

# Course:CS 2302 MW 1:30-2:50, Author:David Ayala  
# Assignment:Lab #5, Instructor: Olac Fuentes  
# Teaching Assistant: Maliheh Zargaran, Date of last Modification: 4/3/2019  
# Purpose of program: Prompt the user to choose a table implementation (binary search tree or hash table with chaining).  
# 2. Read the file ”glove.6B.50d.txt” and store each word and its embedding in a table with the chosen  
# implementation.  
# 3. Compute and display statistics describing your hash table. See the appendix for examples for both  
# implementations.  
# 4. Read another file containing pairs of words (two words per line) and for every pair of words find and display  
# the ”similarity” of the words.  
# 5. Display the running times required to build the table (item 2) and to compute the similarities (item 4).  
  
import math  
import time  
  
class BST(object):  
 # Constructor  
 def \_\_init\_\_(self, item, left=None, right=None):  
 self.item = item  
 self.left = left  
 self.right = right  
  
  
def InsertO(T, newItem):  
 if T == None:  
 T = BST(newItem)  
 elif T.item.word > newItem.word:  
 T.left = InsertO(T.left, newItem)  
 else:  
 T.right = InsertO(T.right, newItem)  
 return T  
  
  
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 Delete(T, del\_item):  
 if T is not None:  
 if del\_item < T.item:  
 T.left = Delete(T.left, del\_item)  
 elif del\_item > T.item:  
 T.right = Delete(T.right, del\_item)  
 else: # del\_item == T.item  
 if T.left is None and T.right is None: # T is a leaf, just remove it  
 T = None  
 elif T.left is None: # T has one child, replace it by existing child  
 T = T.right  
 elif T.right is None:  
 T = T.left  
 else: # T has two chldren. Replace T by its successor, delete successor  
 m = Smallest(T.right)  
 T.item = m.item  
 T.right = Delete(T.right, m.item)  
 return T  
  
  
def InOrder(T):  
 # Prints items in BST in ascending order  
 if T is not None:  
 InOrder(T.left)  
 print(T.item, end=' ')  
 InOrder(T.right)  
  
  
def InOrderD(T, space):  
 # Prints items and structure of BST  
 if T is not None:  
 InOrderD(T.right, space + ' ')  
 print(space, T.item)  
 InOrderD(T.left, space + ' ')  
  
def Smallest(T):  
 # Returns smallest item in BST. Error if T is None  
 if T.left is None:  
 return T  
 else:  
 return Smallest(T.left)  
  
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[0] == k:  
 return T  
 if T.item[0] < k:  
 return Find(T.right, k)  
 return Find(T.left, k)  
  
def height(T):  
 if T is None:  
 return 0  
 else:  
 ldepth = 1 + height(T.left)  
 rdepth = 1 + height(T.right)  
 if ldepth < rdepth:  
 return rdepth  
 else:  
 return ldepth  
  
  
def NumberOfNodes(T):  
 if T is None:  
 return 0  
 return 1 + NumberOfNodes(T.left) + NumberOfNodes(T.right)  
  
class HashTableC(object):  
 # Builds a hash table of size 'size'  
 # Item is a list of (initially empty) lists  
 # Constructor  
 def \_\_init\_\_(self, size, num\_items):  
 self.item = []  
 for i in range(size):  
 self.item.append([])  
 self.num\_items = num\_items  
  
def InsertC(H, k, l):  
 # Inserts k in appropriate bucket (list)  
 # Does nothing if k is already in the table  
 if (H.num\_items / len(H.item) >= 1):  
 temp = HashTableC((len(H.item) \* 2) + 1, 0)  
 for i in range(len(H.item)):  
 for j in range(len(H.item[i])):  
 InsertC(temp, H.item[i][j][0], H.item[i][j][1])  
 H.item = temp.item  
 H.num\_items = temp.num\_items;  
 b = hashing(k, len(H.item))  
 H.num\_items += 1  
 H.item[b].append([k, l])  
  
  
def load\_fact(H):  
 load = 0  
 for i in range(len(H.item)):  
 if H.item[i] != []:  
 for j in range(len(H.item[i])):  
 load += len(H.item[i])  
 load = load / len(H.item)  
 return load  
  
  
def FindC(H, k):  
 # Returns bucket (b) and index (i)  
 # If k is not in table, i == -1  
 b = hashing(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 FindCSecond(H, k):  
 # Returns bucket (b) and index (i)  
 # If k is not in table, i == -1  
 b = hashing(k, len(H.item))  
 for i in range(len(H.item[b])):  
 if H.item[b][i][0] == k:  
 return H.item[b][i][1]  
 return b, -1, -1  
  
  
def hashing(s, n):  
 r = 0  
 for c in s:  
 r = (r \* 27000 + ord(c)) % n  
 return r  
  
  
def percentageH(H):  
 empty = 0  
 for i in range(len(H.item)):  
 if H.item[i] == []:  
 empty += 1  
 percentage = (empty \* 100) / len(H.item)  
 return percentage  
  
class word(object):  
 def \_\_init\_\_(self, word, numbers):  
 self.word = word  
 self.numbers = numbers  
  
def HashSimilarity(item1, item2):  
 dot = 0  
 for i in range(50):  
 dot += item1[i] \* item2[i]  
  
 mag1 = 0  
 for i in range(50):  
 mag1 += item1[i] \* item1[i]  
 mag1 = math.sqrt(mag1)  
 mag2 = 0  
 for i in range(50):  
 mag2 += item2[i] \* item2[i]  
 mag2 = math.sqrt(mag2)  
  
 return dot / (mag1 \* mag2)  
  
def TreeSimilarity(item1, item2):  
 dot = 0  
 for i in range(50):  
 dot += item1.item[1][i] \* item2.item[1][i]  
  
 mag1 = 0  
 for i in range(50):  
 mag1 += item1.item[1][i] \* item1.item[1][i]  
 mag1 = math.sqrt(mag1)  
 mag2 = 0  
 for i in range(50):  
 mag2 += item2.item[1][i] \* item2.item[1][i]  
 mag2 = math.sqrt(mag2)  
  
 return dot / (mag1 \* mag2)  
  
def StandardDeviation(H):  
 length = 0  
 for i in range(len(H.item)):  
 length += len(H.item[i])  
 mean = length / len(H.item)  
 total = 0  
 for i in range(len(H.item)):  
 temp = len(H.item[i]) - mean  
 total += (temp \* temp)  
 amount = total / len(H.item)  
 amount = math.sqrt(amount)  
 return amount  
  
file = open('glove.6B.50d.txt', encoding='utf-8')  
  
array = []  
  
for line in file:  
 string = file.readline()  
 strsplit = string.split()  
 temp = strsplit[0]  
 temp2 = strsplit[1:]  
 temp3 = []  
 for i in range(50):  
 temp3.append(float(temp2[i]))  
 if temp.isalpha():  
 array.append([temp, temp3])  
  
select = input('Please select either Binary Search Tree (1) or for Hash Table (2):')  
  
if select is '1':  
 start = time.time()  
 T = None  
 print('Selected binary search tree, please wait while it builds.')  
 for i in range(len(array)):  
 T = Insert(T, array[i])  
 end = time.time()  
 print('Height of tree:')  
 print(height(T))  
 print('Number of nodes in tree:')  
 print(NumberOfNodes(T))  
 print('Running time for binary search tree in Seconds:')  
 print(round((end - start), 5))  
 print('determining similarities in words.txt')  
  
 text = open('words.txt', encoding='utf-8')  
 for line in text:  
 string = text.readline()  
 split = string.split()  
 if len(split) == 2:  
 temp = Find(T, split[0])  
 temp2 = Find(T, split[1])  
 if temp is not None and temp2 is not None:  
 print(split)  
 print('=')  
 print(TreeSimilarity(temp, temp2))  
  
  
if select is '2':  
 start = time.time()  
 Size = 11  
 H = HashTableC(Size, 0)  
 print('Selected, building hash table with chaining, please wait while it builds.')  
 for i in range(len(array)):  
 InsertC(H, array[i][0], array[i][1])  
 end = time.time()  
 print('Load Factor:')  
 print(round(load\_fact(H), 4))  
 print('Table size:')  
 print(len(H.item))  
 print('Standard deviation:')  
 print(round(StandardDeviation(H), 4))  
 print('Percentage of empty lists:')  
 print(round(percentageH(H), 4))  
 print('Running time for hash table in seconds:')  
 print(round((end - start), 5))  
 print('determining similarities in words.txt')  
  
 text = open('words.txt', 'r')  
 for line in text:  
 string = text.readline()  
 split = string.split()  
 if len(split) == 2:  
 temp = FindCSecond(H, split[0])  
 temp2 = FindCSecond(H, split[1])  
 if temp is not None and temp2 is not None:  
 print(split)  
 print(' = ')  
 print(HashSimilarity(temp, temp2))