Nancy Hernandez

Olac Fuentez

Lab 5 Report

**Introduction**

Purpose of this lab was to compare the running times of two implementations of tables to retrieve word embeddings to enable the (hopefully fast) comparison of two given words.

**Proposed Solution**

First, I asked the user whether they want to make a binary search tree or a hash table. If The pick one, then I begin by making the binary search tree. If they choose number two then I go ahead and create the hash table with chaining.

***Binary Search Tree*-**

I bigin by reading the file “glove.6B.50d.txt”. I do this with a for loop that splits the word from the embeddings. Once ive successfully split them then I insert them in my tree. I do time how long it takes to read the file and insert each word and its embeddings. Once this is done, I obtain the height of the tree and print it. I print the running time, and the number of nodes which I obtain by using a counter inside the for loop.

***Hash Table with Chaining*-**

When it comes to creating the hash table it is like the tree method, but the difference is that you need to determine the initial table size, which in this case I decided to use 31 because you its best to use a prime number. I created a temporary word that stores the sting and the embeddings. I then inserted this temporary word into the table. Again, I time how long it took to finish this process. I printed the initial and final size, the percent empty, the time, the number of nodes and the standard deviation.

Once I have finished creating the tree and the hash table, I moved on to reading the second file. I created two different methods as the tree and the table both have their different necessities. For the binary tree method, I began with reading the file and once again splitting both words and once those words are split, I send them to my Find method where I am to get the corresponding embeddings. For the hash table everything is the same as the binary method except that I send the words to the FindC method provided. And for both methods I time how long it takes to complete these tasks.

To find the empty percentage in the hash table I created a for loop that checks if the length of each bucket is less than one, and if it is it increases the counter. Then to obtain the percentage I divide the counter by the length of the table and multiply it by 100.

To obtain the standard deviation I create an empty list and create a for loop that appends the items in the table to the empty list I created. Then I get the deviation by using the python function np.std.() that does the work for me.

**Setup**

To complete this lab I used an HP Pavilion x360 Convertible with a 2.71 GHz Intel® Core(TM) i5 processor.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**Results**

**Time Analysis**

I was unable to get times for the similarities as I couldn’t figure out why my code wasn’t working. Although I was able to time how long it took to read the first file and create the binary search tree and hash table using chaining.

Time it took to read file and create binary tree: 4.578124284744263 seconds **O(n)**

Time it took to read and create hash table with chaining: 7.158051490783691 seconds **O(n)**

**Conclusion**

From this lab I concluded that the Hash Table using Chaining is significantly faster than the Binary Search Tree as it goes through the data, the words in this case, much faster due to the time.

**Appendix**

# 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\*n + 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))

**Academic Dishonesty Statement**

I, Nancy Hernandez, was not involved in any copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data in laboratory reports. Neither did I participate in any type of collusion involving collaboration with another person to commit an academically dishonest act.