Lab 05 Report

In lab 5 we are instructed to implement hash tables with linear probing to retrieve word embeddings to enable the comparison of two given words. We are given several hash functions. And our task is to determine which is the best for creating and for this task. we are given a text file with a large amount of words, and we need to store each one of the words, along with its embeddings into the table with the word being used as a hashing key. For easy retrieval from the table, the two will be stored into a node and the different nodes will be stored into the table. We will double the size of the hash table every time the load factor reaches 0.5. We are also instructed to prompt the user to choose a hash function to evaluate.

The idea I initially thought of was to create a function for every different hashing function. This eventually became quite messy, so I thought that I could use one insert function that would have if statements that would check for user input and would use the specified hashing method. I did this using the global variable choice that would change upon user input. This was also done for the retrieve function that would need to be using the same hashing method to retrieve the word embeddings so that the same hashing method is used to insert and retrieve.

The first function we are asked to implement, is the length of the string % n. this was relatively simple considering the fact that we have mainly used this function in previous hash table exercises. Although, we were never given a hash table with big enough to realize how slow this function would actually be. the function is really slow because of the amount of collisions that occur during insertion. It also becomes really slow when the same hash function is being used with the retrieval of the words. There are so many collisions that occur, that the retrieval of the word is not going to be as smooth and efficient as expected.

The second function we were asked to implement is just a tad bit faster. But is also very slow with the large amounts of data that we are using. We are asked to implement the ascii value of the first character in the string % n. computing the ascii value of the first character in the string adds quite a bit of space in the table. Although there are still quite a bit of collisions causing the function to still run quite slow. The same function is used when retrieving word to add more accuracy to the times being tested.

A faster hashing function we are asked to implement, is the product of the ascii values of the fist and last characters in the string. The function is a lot faster than the first two functions. There is a lot more space in the table and less collisions are happening as a result. This will also cause my retrieve functions to be a lot more efficient, considering that there are less collisions occurring. Building the function becomes faster and faster as we are using better hash functions and much easier to implement.

The next function we are asked to implement is the sum of the ascii values of the characters in the string. The biggest difference with this function and the previous ones is that I needed to loop through all the characters in the string and find the sum of all of them. I can then use the sum variable that I created, the same way that I created all of the previous functions. The running time of this function was much faster than all of the previous functions. And much less collisions occurred during the insertion process as well as the retrieving function.

The following function is the recursive function we are asked to implement. The function very fast. The function needed to be implemented differently. I created a separate function for this one where the recursion would take place. The function given to us was as follows… h(”,n) = 1; h(S,n) = (ord(s[0]) + 255\*h(s[1:],n))% n. I set my base case to check if the length of s was equal to one and once it was, I would return. And I would then use the recursive equation to finish. Inside the insert function, I would call the recursive function and insert the same way I inserted the rest of the functions. The function is much faster and more efficient causing less chaining and bigger table.

The last function asked to implement was our own. I decided to do a very similar function as the rest of the functions. I used the length of the string multiplied by 100 and divided by 2 % length of the string. I tried to cause as little chaining as possible by making the words string a lot bigger. This function was not much better than the second and third functions but still did not work as fast as it really should

The next step in the lab was to read another file containing pairs of words, and for every pair of words we will find and display the similarities of the two words. To compute the similarities of the two words we will compare the embeddings of the two words with the sim function. This was done by creating another function ‘sim’ which receives embedding 0 and embedding 1 finding the dot product of the two arrays, also using another function which computed the magnitude to use for the denominator. This helped compare the embeddings, which made it an easier process.

When reading the word file would read the file and split each word. Both words will get their similarities using the ‘sim’ function. The word file is called in the main to compare each word in the file after the chosen insertion function is used. Before the comparisons are done, the stats and running times will be computed.

The table size is done by returning the length of the table once all insertions have been done. The load factor is computed by dividing the length of H.num\_items by lenth of the string. The longest chain needs to be done with a separate function that iterates through the whole list and counts the amount of times there is a word in place. The challenging part of the function is if there is a chain going from the end of the list to the beginning. The way I went about this was by doubling the size of the string just incase of this situation. The running time is simply done with a time of the functions being computed.

I ran every hash function with the range of 10000 to test my times because if I make a table the of the size of all the words the running times would take too long. The usual time taken to retrieve data from the table was usually around 7 seconds.

The following is a line graph describing the build times of the function. Noticeably, the build time decreases from the first hash function to the fifth. But also goes up very little with the function that I chose to implement.

#Daniel Solorzano

#Data Structures

#Fall 2019

import numpy as np

import time

import math

import sys

choice = 1

class HashTableL(object):

#builds a hash table of size 'size'

def \_\_init\_\_ (self, size, num\_items=0):

self.item = [None] \* size

self.num\_items = 0

class Word:

def \_\_init\_\_(self, word, embedding):

self.word = word

self.embedding = embedding

def getWords(filename):

words = []

embeddings = []

file = open((filename), 'r', encoding = 'utf8')

print("Retriving data from the file:" , filename, ".....")

start = time.time() # start timer

# importing words and embeddings from file

for line in file:

splitLine = line.split(sep = ' ')

word = splitLine[0]

words.append(word)

embedding = np.array([float(val)for val in splitLine[1:]])

embeddings.append(embedding)

end = time.time() # end timer

total = end - start

print("Finished retrieving data.....")

print("Time taken:", total)

return words, embeddings

def get2Words(file):

print("Reading word file to determine similarities")

print("\nWord similarities found")

file = open(file, 'r', encoding = 'utf8')

for line in file:

splitLine = line.strip().split(sep = ' ')

word1 = splitLine[0]

word2 = splitLine[1]

r1 = retrieve(word1)

r2 = retrieve(word2)

print("Similarity [", word1,",", word2,"] =", sim(r1, r2))

## Insert method

def Insert(H, w, e):

sum = 0

word = Word(w, e) # create the object with the word w

for i in w:

asci = ord(i)

sum = sum + asci

#print('Ascii val:', sum)

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

if choice is 1:

pos = (len(w) + i) % len(H.item)

if H.item[pos] == None:

H.item[pos] = word

H.num\_items += 1

return None

if choice is 2:

pos = (ord(w[0]) + i) % len(H.item)

if H.item[pos] == None:

H.item[pos] = word

H.num\_items += 1

return None

if choice is 3:

pos = (ord(w[0]) \* ord(w[-1]) + i) % len(H.item)

if H.item[pos] == None:

H.item[pos] = word

H.num\_items += 1

return None

if choice is 4:

pos = (sum + i) % len(H.item)

if H.item[pos] == None:

H.item[pos] = word

H.num\_items += 1

return None

if choice is 5:

pos = recursiveH(w, len(H.item))

if H.item[pos] == None:

H.item[pos] = word

H.num\_items += 1

return None

if choice == 6:

pos = int((len(w)\*100/2 + i) % len(H.item))

if H.item[pos] == None:

H.item[pos] = word

H.num\_items += 1

return None

return None

def recursiveH(S, n):

if len(S) == 1:

return 1

else:

return(ord(S[0]) + 255 \* recursiveH(S[1:],n) ) % n

def retrieve(word):

if choice == 1:

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

pos = (len(word)+i) % len(H.item)

if H.item[pos] == None:

pass

else :

if H.item[pos].word == word:

return H.item[pos].embedding

print("word was not found")

if choice == 2:

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

pos = (ord(word[0]) + i) % len(H.item)

if H.item[pos] == None:

pass

else :

if H.item[pos].word == word:

return H.item[pos].embedding

print("word was not found")

if choice == 3:

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

pos = (ord(word[0])\* ord(word[-1]) + i) % len(H.item)

if H.item[pos] == None:

pass

else :

if H.item[pos].word == word:

return H.item[pos].embedding

print("word was not found")

if choice == 4:

sum = 0

for i in word:

asci = ord(i)

sum = sum + asci

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

pos = (sum + i) % len(H.item)

if H.item[pos] == None:

pass

else :

if H.item[pos].word == word:

return H.item[pos].embedding

print("word was not found")

if choice == 5:

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

pos = (ord(word[0])+ ord(word[-1]) + i) % len(H.item)

#print("Pos",pos,"Word", word)

if H.item[pos] == None:

pass

else :

if H.item[pos].word == word:

return H.item[pos].embedding

print("word was not found")

if choice == 6:

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

pos = int((len(word)\*100/2 + i) % len(H.item))

if H.item[pos] == None:

pass

else :

if H.item[pos].word == word:

return H.item[pos].embedding

print("word was not found")

def mag(x):

return math.sqrt(sum(i\*\*2 for i in x))

def sim(e0, e1):

numer = np.dot(e0,e1)

denom = mag(e0)\* mag(e1)

return numer/denom

def longestChain(H):

count = 0

temp = 0

for i in range(len(H.item)\*2):

pos = (i) % len(H.item)

if H.item[pos] != None :

temp +=1

else :

if temp > count:

count = temp

temp = 0

return count

if \_\_name\_\_ == "\_\_main\_\_":

words, embeddings = getWords("glove.6b.50d.txt")

H = HashTableL(101) # set initial hash table size

choice = int(input("Choose hash function 1 to 6 \n"))

if choice > 6 or choice < 1:

print("You chose a number that was not a hash function.... Aborting")

sys.exit()

print("Choice:", choice)

print("Building hash table\n")

start = time.time()

for i in range(10000):

LF = H.num\_items / len(H.item)

if LF >= 0.5: # increase table size if load factor is greater than 0.5

#H.num\_items = (H.num\_items\*2)+1

#print("Load factor reached, doubling table size to:", len(H.item)\*2)

#start = time.time() # start timer

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

H.item.append(None)

Insert(H, words[i], embeddings[i])

end = time.time() # end timer

total = end - start

print("Hash table stats:")

print("Table size:", len(H.item))

print("Load Factor:", H.num\_items / len(H.item))

print("Longest chain:", longestChain(H))

print("Running time for table construction:",total)

print()

# print stats

start = time.time() # start timer

get2Words('words.txt')

end = time.time() # start timer

total = end - start

print("Running time for query processing:" , total)

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.

X\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_