Summer 2019

Laboratory 4 –Hash table

CS 2302 – Data structures summer 2019

Dilan Ramirez

INSTRUCTOR: OLAC FUENTES

TEACHING ASSISTANTS (TA): ISMAEL VILLANUEVA-MIRANDA

AUGUST 5, 2019

Contents

[Introduction 2](#_Toc15996822)

[Implementation 2](#_Toc15996823)

[Experimental results 3](#_Toc15996824)

[Conclusion 7](#_Toc15996825)

[Appendix 7](#_Toc15996826)

[Honesty Certification 12](#_Toc15996827)

# Introduction

Natural Language Processing (NLP) is the sub-field of artificial intelligence that deals with designing algorithms, programs, and systems that can understand human languages in written and spoken forms. Word embeddings are a recent advance in NLP that consists of representing words by vectors (or arrays) of floating point numbers in such a way that if two words are similar, their embeddings are also similar.

The task for this lab is to implement hash tables with linear probing to retrieve word embeddings to enable the (hopefully fast) comparison of two given words. You will compare several hash functions and determine which is the best for this task. Implement and compare the following hash functions, where n is the length of the table:

1. The length of the string % n
2. The ascii value (ord(c)) of the first character in the string % n
3. The product of the ascii values of the first and last characters in the string % n
4. The sum of the ascii values of the characters in the string % n
5. The recursive formulation h(”,n) = 1; h(S,n) = (ord(s[0]) + 255\*h(s[1:],n))% n
6. Another function of your choice

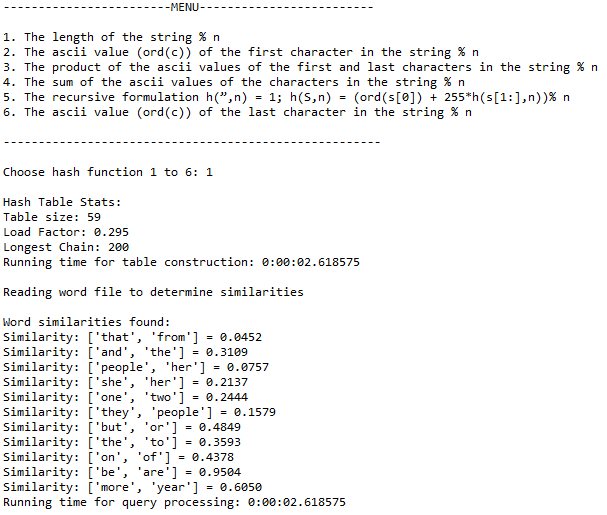
# Implementation

The first step was to read the text file which contains all the words and embeddings. To do so, I use my function called “readFile.” This function first open the file and creates a list with all the file’s content. After that, the list I split the list in every comma because the word and its embedding is separated by a comma. Then, I call my function “convert”. This function will create a list and convert its elements into strings. After that, I remove the first element of each list that is the word to convert the rest elements of the list to float. Finally, I appended the string with the float embeddings. In the same function I assign the string and the embeddings to the object’s parameters.

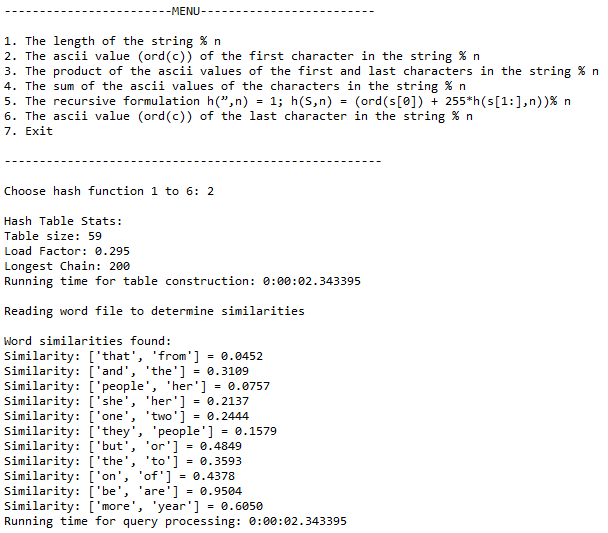
Now I use the function “Insert\_By\_Positions” to insert the address of the object “slot” which contains the word and the embeddings. The position of the slot in the hash table is given by the length of the word mod the length of the hash table. Once the position is found, the object address of that word is stored in that position. To keep track of the load factor of the table, I used a variable that is increment by one every time a word is inserted in the hash table. If the load factor is equal to 0.50, I used my function “resizeHashTable” and “checkHashTableSize” to increment the size of the table by two plus one.

# Experimental results

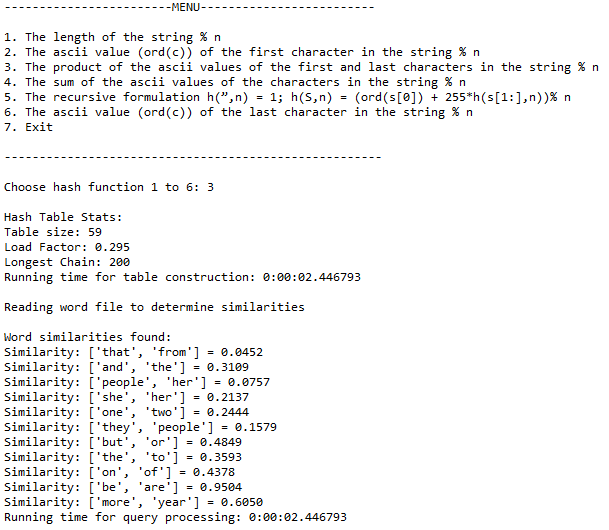
1. The length of the string % n



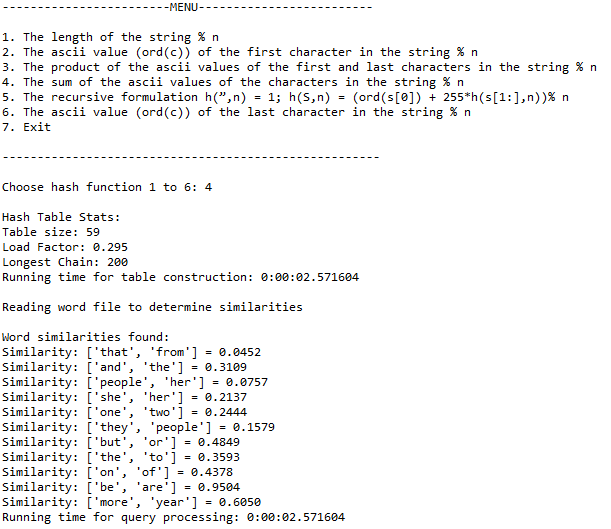
1. The ascii value (ord(c)) of the first character in the string % n



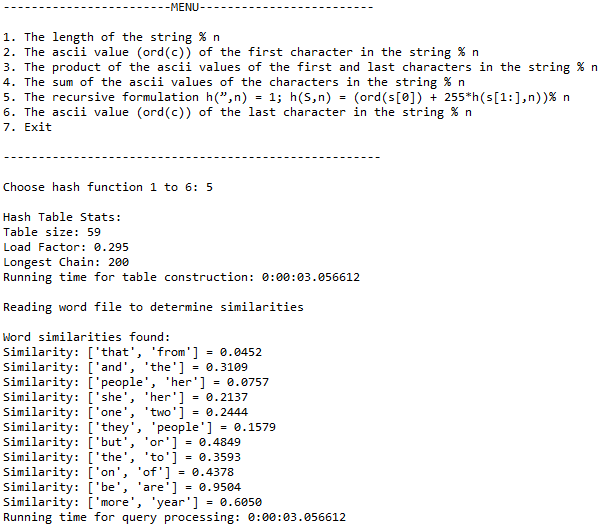
1. The product of the ascii values of the first and last characters in the string % n



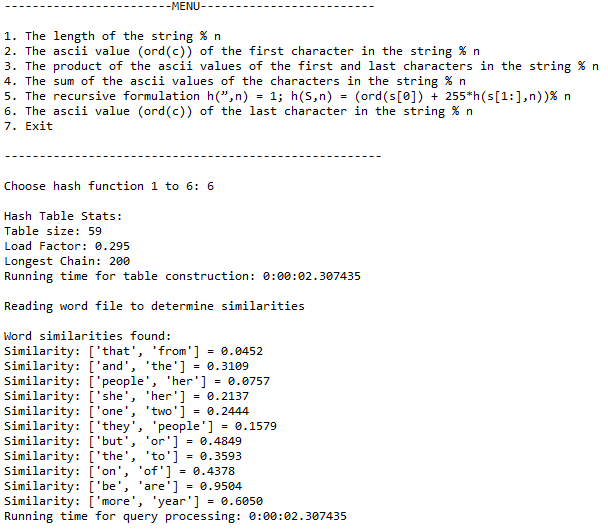
1. The sum of the ascii values of the characters in the string % n



1. The recursive formulation h(”,n) = 1; h(S,n) = (ord(s[0]) + 255\*h(s[1:],n))% n



1. The ascii value (ord(c)) of the last character in the string % n"



# Conclusion

By doing this lab, I learned how implement a hash table in a real-life problem. I also learned how to read a txt file, and how to order all its content. I had many problems in this lab but overcome them until I found the insert problem. The problem was that the object is not inserted in the hash table after doing it a few times as the picture from above shows it.

# Appendix

import numpy as np

from datetime import datetime

start\_time = datetime.now()

import math

class HashTable(object):

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

self.item = list(np.zeros(size,dtype=np.int)-1)

self.num\_items = num\_items

class Slot(object):

def \_\_init\_\_(self, word=" ", embedding=np.array(50)):

self.word = word

self.embedding = embedding

def loadFactor(H):

return float(H.num\_items / len(H.item))

def longestChain(H):

neg\_ones=np.argwhere(H.item == -1)

if len(neg\_ones)==0:

return len(H.item)

neg\_ones=np.append(neg\_ones, neg\_ones[0]+len(H.item))

print('Negative ones:',neg\_ones)

chain\_lens = neg\_ones[1:]-neg\_ones[:-1]-1

print('Chain lengths:',chain\_lens)

return np.max(chain\_lens)

def readFile(file):

words\_embeddings = []

with open(file, 'r') as f:

for line in f:

line = line.strip('\n')

line = line.split(' ')

if line[0].isalpha():

S = Slot(line[0])

S.embedding = np.array(list(map(float, line[1:])))

words\_embeddings.append(S)

return words\_embeddings

def resizeHashTable(H, newH, option):

L = list(H.item)

while (-1) in L:

L.remove(-1)

for i in range(len(L)):

Insert\_By\_Positions(newH, L[i], option)

H = newH

def Insert\_By\_Positions(H, S, option):

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

if option == 1:

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

if option == 2:

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

if option == 3:

product = (ord(S.word[0]) \* ord(S.word[-1]))

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

if option == 4:

sumascii = 0

for letter in S.word:

sumascii += ord(letter)

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

if option == 5:

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

if option == 6:

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

if H.item[pos] == -1:

H.item[pos] = S

H.num\_items += 1

return pos

return -1

def checkHashTableSize(file, option):

H = HashTable(200)

for i in range(len(file)):

Insert\_By\_Positions(H, file[i], option)

if H.num\_items >= int(len(H.item) \* 0.5):

newHashTable = HashTable((2 \* len(H.item)) + 1)

resizeHashTable(H, newHashTable, 1)

H = resizeHashTable

return H

def tableSize(H):

num\_items = 0

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

if H.item[i] != -1:

num\_items += 1

return num\_items

def HashTableInfo(H):

print('\nHash Table Stats:')

print('Table size:', tableSize(H))

print('Load Factor:', round(loadFactor(H), 6))

print('Longest Chain:', longestChain(H))

return

def print\_HashTable\_Values(H):

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

if H.item[i] != -1:

print(H.item[i].word)

print(H.item[i].embedding)

return

def menu():

print("\n------------------------MENU-------------------------")

print("\n1. The length of the string % n")

print("2. The ascii value (ord(c)) of the first character in the string % n")

print("3. The product of the ascii values of the first and last characters in the string % n")

print("4. The sum of the ascii values of the characters in the string % n")

print("5. The recursive formulation h(”,n) = 1; h(S,n) = (ord(s[0]) + 255\*h(s[1:],n))% n")

print("6. The ascii value (ord(c)) of the last character in the string % n")

print("\n------------------------------------------------------")

return

def find\_By\_Position(H, word, option):

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

if option == 1:

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

if option == 2:

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

if option == 3:

product = (ord(word[0]) \* ord(word[-1]))

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

if option == 4:

sumascii = 0

for letter in word:

sumascii += ord(letter)

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

if option == 5:

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

if option == 6:

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

if H.item[pos] != -1:

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

return H.item[pos].embedding

return

def word\_Similarities(H, word, option):

magnitude1,magnitude2 = 0, 0

dot1, dot2, sim = 0, 0, 0

with open('similar2.txt', 'r') as sims:

for line in sims:

line = line.strip('\n')

line = line.split(' ')

firstWord = line[0]

secondWord = line[1]

#print(firstWord)

word1 = find\_By\_Position(H, firstWord, option)

word2 = find\_By\_Position(H, secondWord, option)

for num in word1:

magnitude1 += num\*num

magnitude1 = math.sqrt(magnitude1)

for num2 in word2:

magnitude2 += num2\*num2

magnitude2 = math.sqrt(magnitude2)

for i in word1:

for j in word2:

dot1 = i\*j

dot2 += dot1

sim = dot2/abs(magnitude1\*magnitude2)

print("Similarity:", [firstWord, secondWord], '=', "{:.4f}".format(abs(sim)))

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

file = readFile("words\_embeddings.txt")

menu()

option = 0

while option < 7:

option = int(input("Choose hash function 1 to 6: "))

if option > 6:

print("BYE!")

if option > 0 and option < 7:

if option == 1:

H = checkHashTableSize(file, option)

HashTableInfo(H)

end\_time = datetime.now()

total\_time = end\_time - start\_time

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

print("\nReading word file to determine similarities")

print("\nWord similarities found:")

word\_Similarities(H,'',option)

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

if option == 2:

H = checkHashTableSize(file,option)

HashTableInfo(H)

end\_time = datetime.now()

total\_time = end\_time - start\_time

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

print("\nReading word file to determine similarities")

print("\nWord similarities found:")

word\_Similarities(H,'',option)

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

if option == 3:

H = checkHashTableSize(file,option)

HashTableInfo(H)

end\_time = datetime.now()

total\_time = end\_time - start\_time

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

print("\nReading word file to determine similarities")

print("\nWord similarities found:")

word\_Similarities(H,'',option)

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

if option == 4:

H = checkHashTableSize(file,option)

HashTableInfo(H)

end\_time = datetime.now()

total\_time = end\_time - start\_time

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

print("\nReading word file to determine similarities")

print("\nWord similarities found:")

word\_Similarities(H,'',option)

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

if option == 5:

H = checkHashTableSize(file,option)

HashTableInfo(H)

end\_time = datetime.now()

total\_time = end\_time - start\_time

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

print("\nReading word file to determine similarities")

print("\nWord similarities found:")

word\_Similarities(H,'',option)

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

if option == 6:

H = checkHashTableSize(file,option)

HashTableInfo(H)

end\_time = datetime.now()

total\_time = end\_time - start\_time

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

print("\nReading word file to determine similarities")

print("\nWord similarities found:")

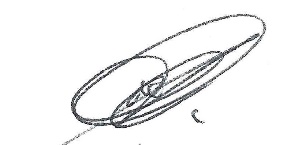
word\_Similarities(H,'',option)

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

# 

# Honesty Certification

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 provide inappropriate assistance to any student in the class.

 08/ 05 / 2019

Dilan Ramirez Date