PAOLA TERRAZAS

CS2302

FINDING SIMILARITIES

DR. FUENTES

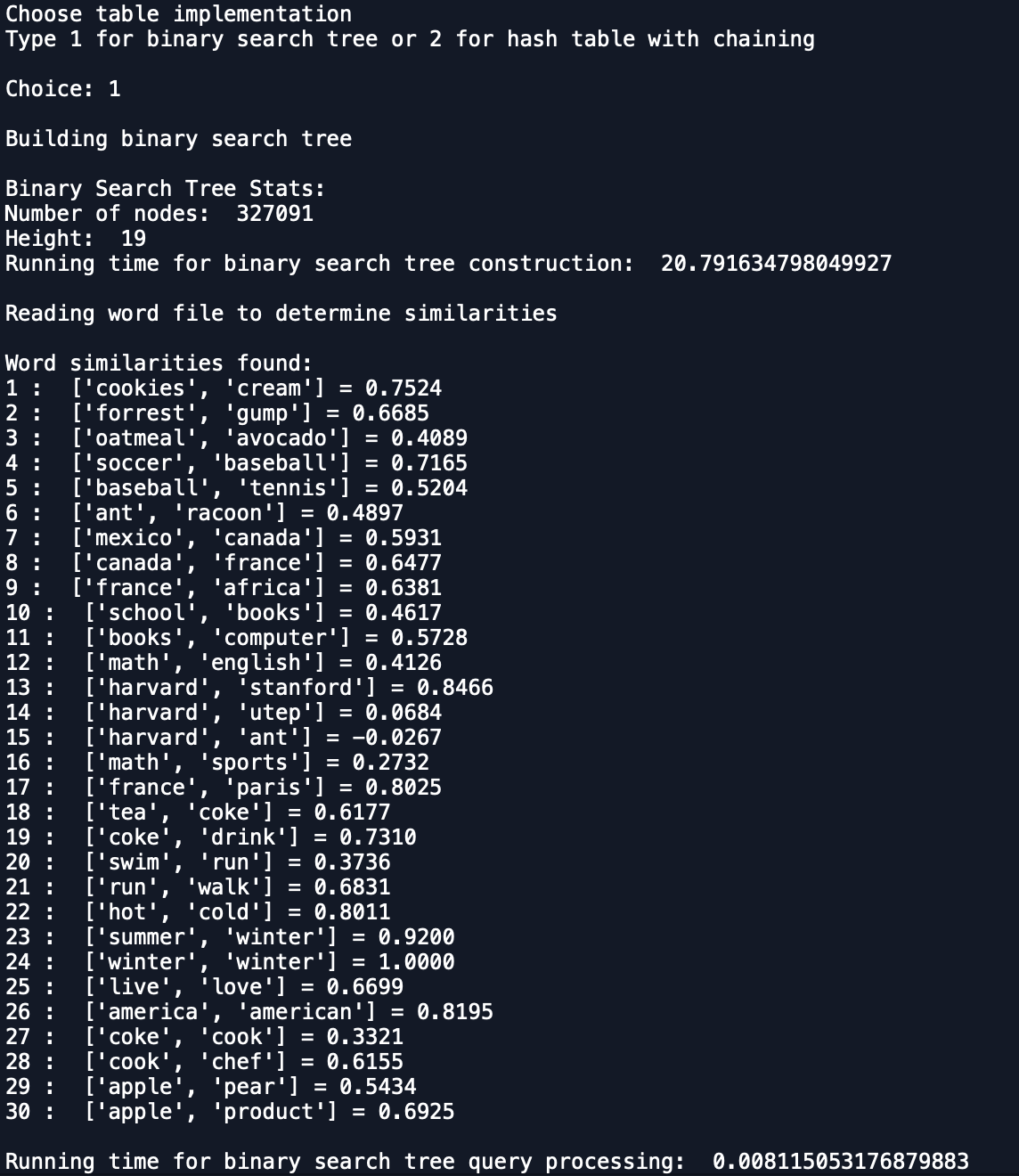
TA – ANINDITA NATH

4.01.2019

LAB 5 REPORT

This lab was about Natural Language Processing (NLP). It is a sub-filed of artificial intelligence that deals with designing algorithms, programs and systems. Our task was to compare running times for two implementations of tables to retrieve word embeddings to enable the comparison of two given words. We were told to use a binary search tree and a hash table with chaining. I know binary search trees very well, therefore working with option 1 was not difficult. I was able to use the glove file and write a function to find the similarities. I created a separate file “words.txt” with 30 pairs of words to be compared. I also imported time to get the program execution times. The point is to use these two different data structures to get the output. For option 2, I had trouble. I could not figure out how to do exactly what the assignment asked using hash table with chaining considering it is a different data structure than the one I’m most familiar with. Although, I was able to understand the main purpose of using different strategies for understanding time complexity when using very large files. I also used word embedding as a type of word representation. There was two files used, one that contains the words and embeddings and the other that contained all the words to be compared.

Following is my results when choosing option 1, binary search tree for the comparisons.



**# SOURCE CODE**

# PAOLA TERRAZAS

# CS2302

# Finding similarities

# DR. FUENTES

# TA - ANINDITA NATH

# 4.01.2019

# create a hash table and binary search tree to read from glove file to search words and similarities as well as other functions

import numpy as np

import math

import time

# Hash Table

# Implementation of hash tables with chaining

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):

# Inserts k in appropriate bucket (list)

# Does nothing if k is already in the table

b = k%len(H.item)

H.item[b].append(k)

def FindC(H,k):

# Returns bucket (b) and index (i)

# If k is not in table, i == -1

b = k%len(H.item)

try:

i = H.item[b].index(k)

except:

i = -1

return b, i

def DeleteC(H,k):

# Returns k from appropriate list

# Does nothing if k is not in the table

# Returns 1 in case of a successful deletion, -1 otherwise

b = k%len(H.item)

try:

H.item[b].remove(k)

return 1

except:

return -1

# BST

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

def LtoT(T,n):

if len(n) == 0:

return None

mid = len(n)//2

if T is None:

mid1 = BST(n[mid])

mid1.right = LtoT(T,n[mid+1:])

mid1.left = LtoT(T,n[:mid])

return mid1

def numofNodes(T):

if T is None:

return 0

return 1 + numofNodes(T.left) + numofNodes(T.right)

def getHeight(T):

if T is None:

return 0

left = 1+getHeight(T.left)

right = 1+getHeight(T.right)

if left > right:

return left

return right

def compare(word1,word2):

if word1 > word2:

return True

return False

def findNode(T,k):

if T is None:

return -1

if T.item[0] == k:

return T.item

i = T.item[0]

if compare(i,k):

return findNode(T.left,k)

return findNode(T.right,k)

def simi(T,word,word2):

word = findNode(T,word)

word2 = findNode(T,word2)

dot = dotPro(word,word2)

mag = magnitude(word)

mag2 = magnitude(word2)

total = mag \* mag2

return dot/total

def magnitude(w):

m = 0

for i in range(len(w[1])):

m += w[1][i] \* w[1][i]

return math.sqrt(m)

def dotPro(word,word2):

dot = 0

for i in range(len(word[1])):

dot += word[1][i] \* word2[1][i]

return dot

def filetoA(file):

f = open(file, encoding="utf8")

A = f.readlines()

f.close

return A

def Split(n):

w = []

for i in range(len(n)):

s = n[i].split()

w.append(s)

return w

def wordEmbedding(n):

w = []

for i in range(len(n)):

if n[i][0].isalpha():

ls = np.array(n[i][1:])

l = ls.astype(np.float)

s = [n[i][0],l]

w.append(s)

return w

# Main

print("Choose table implementation")

print("Type 1 for binary search tree or 2 for hash table with chaining")

ans = input("Choice: ")

ans = int(ans)

if ans == 1:

txt = filetoA('words.txt')

txt = Split(txt)

print()

print("Building binary search tree")

print()

start = time.time()

T = None

filename = 'glove.6B.50d.txt'

inArr = filetoA(filename)

splitArray = Split(inArr)

emb = wordEmbedding(splitArray)

emb.sort()

T = LtoT(T,emb)

end = time.time()

print("Binary Search Tree Stats:")

print("Number of nodes: ", numofNodes(T))

print("Height: ", getHeight(T))

print("Running time for binary search tree construction: ", end-start)

print()

print("Reading word file to determine similarities")

print()

print("Word similarities found:")

start = time.time()

for i in range(len(txt)):

print(i+1,': ',txt[i],end='')

print(" = ",end='')

print("{0:.4f}".format(simi(T,txt[i][0],txt[i][1])))

end = time.time()

print()

print("Running time for binary search tree query processing: ", abs(start-end))

elif ans == 2:

txt = filetoA('words.txt')

txt = Split(txt)

print()

print("Building hash table with chaining")

print()

start = time.time()

filename = 'glove.6B.50d.txt'

else:

print("Invalid choice, try again.")

**# WORDS.TXT**

cookies cream

forrest gump

oatmeal avocado

soccer baseball

baseball tennis

ant racoon

mexico canada

canada france

france africa

school books

books computer

math english

harvard stanford

harvard utep

harvard ant

math sports

france paris

tea coke

coke drink

swim run

run walk

hot cold

summer winter

winter winter

live love

america american

coke cook

cook chef

apple pear

apple product

Scholastic Dishonesty

Any student who commits an act of scholastic dishonesty is subject to discipline. Scholastic dishonesty includes, but not limited to cheating, plagiarism, collusion, the submission for credit of any work or materials that are attributable to another person.

* **Cheating**
  + Copying form the test paper of another student
  + Communicating with another student during a test
  + Giving or seeking aid from another student during a test
  + Possession and/or use of unauthorized materials during tests (i.e. Crib notes, class notes, books, etc)
  + Substituting for another person to take a test
  + Falsifying research data, reports, academic work offered for credit
* **Plagiarism**
  + Using someone’s work in your assignments without the proper citations
  + Submitting the same paper or assignment from a different course, without direct permission of instructors
* **Collusion**
  + Unauthorized collaboration with another person in preparing academic assignments

Sign \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: 4/03/2019